บทที่ 11

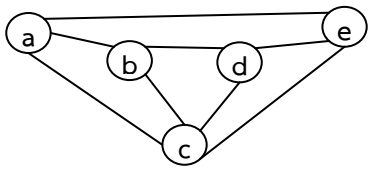
กราฟ

กราฟ (Graph) เป็นโครงสร้างข้อมูลแบบไม่เชิงเส้น (Non-linear Data Structure) เช่นเดียวกับโครงสร้างข้อมูลทรี กราฟเป็นโครงสร้างข้อมูลที่นิยมใช้ในการจัดเก็บข้อมูลและถูกนำไปประยุกต์ใช้ในงานด้านต่างๆ เช่น การหาระยะทางที่สั้นที่สุดจากเมืองหนึ่งไปยังเมืองหนึ่งการคำนวณค่าใช้จ่ายในการเดินสายเชื่อมต่อระบบเครือข่ายคอมพิวเตอร์ หรือการเดินสายโทรศัพท์ รวมถึงการประยุกต์ใช้ในสาขาวิชาปัญญาประดิษฐ์ (Artificial Intelligence) โครงสร้างทางเคมีของแก้วเจียรนัย (Crystals) วงจรไฟฟ้า (Electrical Circuitry) และการวิเคราะห์ของภาษาโปรแกรม เป็นต้น

11.1 นิยามกราฟ

กราฟ (Graph) เป็นโครงสร้างข้อมูลที่ประกอบด้วยเซตของโหนด (Node) หรือจุด (Vertex) และเซตของเส้นเชื่อม (Edge) ดังตัวอย่างในรูปที่ 11.1 โดยมีรายละเอียดดังนี้

- แต่ละจุดยอดจะเก็บค่าข้อมูลหรือสมาชิกของกราฟ
- แต่ละเส้นเชื่อมจะแสดงถึงความสัมพันธ์ระหว่างสองจุดยอดใดๆ โดยอาจเรียกว่า อาร์ก (Arc) แต่ละอาร์กอาจเก็บค่าข้อมูลที่เป็นลักษณะประจำ (Attributes) ของเส้นเชื่อม เช่น ชื่อเส้นเชื่อม ค่าน้ำหนักหรือค่าระยะทางของเส้นเชื่อม เป็นต้น



รูปที่ 11.1 แสดงตัวอย่างกราฟ

จากรูปที่ 11.1 แสดงตัวอย่างกราฟ G = (V, E) โดยกราฟ G ประกอบด้วยเซตของจุดยอด V = {a, b, c, d, e} และเซตของเส้นเชื่อม E = { (a,b), (a,c), (a,e), (b,c), (b,d), (c,d), (c,e), (d,e) }โดยแต่ละเส้นเชื่อม e = (u,v) หมายถึง คู่ลำดับของ u และ v เมื่อ u และ v คือ จุดยอดใดๆ ในกราฟ G ที่มีเส้นเชื่อมโยงถึงกัน คำศัพท์พื้นฐานเกี่ยวกับกราฟ มีดังนี้

- ขนาดของกราฟ คือ จำนวนจุดยอดในกราฟ เช่น จากรูปที่ 11. 1 แสดงตัวอย่างกราฟขนาด 5
- กราฟว่าง (Empty Graph) คือ กราฟที่ไม่มีจุดยอด
- จุดยอดใกล้เคียง (Adjacent Vertices (Nodes)) คือ กลุ่มจุดยอดที่มีเส้นเชื่อมถึงกัน เช่น จากรูปที่ 11.1 กลุ่มจุดยอดใกล้เคียงของจุดยอด a คือ {b, c, e}
- ดีกรี (Degree) ของจุดยอด คือ จำนวนเส้นเชื่อมของจุดยอดนั้นเช่น จากรูปที่ 11. 1 ดีกรีของจุดยอด a มีค่าเท่ากับ 3
- เส้นทาง (Path) คือ กลุ่มของจุดยอด $v_1, v_2, ..., v_k$ ซึ่งเป็นลำดับของจุดยอดที่แสดงเส้นทาง

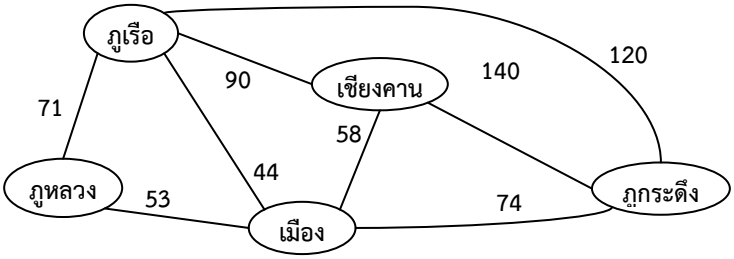
ระหว่างจุดยอด v_1 ถึงจุดยอด v_k และจุดยอด v_i และ v_{i+1} เป็นจุดยอดใกล้เคียงกันหรือเป็นคู่จุดยอดที่มีเส้นเชื่อมระหว่างกัน

- ความยาวของเส้นทางจะเท่ากับจำนวนเส้นเชื่อมในเส้นทาง เช่น เส้นทางระหว่างจุดยอด v_0 ถึงจุดยอด v_n จะมีจำนวนจุดยอดในเส้นทางเท่ากับ n + 1 จุดยอด และมีความยาวของเส้นทางเท่ากับ n เช่น จากรูปที่ 11.1 สามารถแสดงเส้นทาง a, b, d และ e ซึ่งเป็นเส้นทางระหว่างจุดยอด a และจุดยอด e และเป็นเส้นทางขนาด 3 เป็นต้น
- เส้นทางที่ไม่มีจุดยอดซ้ำเลย เรียกว่า เส้นทางแบบง่าย (Simple Path) และหากเป็นเส้นทางแบบง่ายที่มีจุดยอดแรกและจุดยอดสุดท้ายเป็นจุดยอดเดียวกันจะเรียกเส้นทางนั้นว่า ไซเคิล (Cycle) เช่น จากรูปที่ 11.1 สามารถแสดงเส้นทาง a, b, c และ a เป็นเส้นทางในลักษณะของไซเคิลเป็นต้น

(1) ประเภทกราฟตามลักษณะทิศทางของเส้นเชื่อม

ประเภทกราฟตามลักษณะทิศทางของเส้นเชื่อมระหว่างจุดยอด แบ่งเป็น 2 ลักษณะดังนี้

1) กราฟแบบไม่มีทิศทาง (Undirected Graph) คือ กราฟที่ไม่ระบุทิศทางของความสัมพันธ์ระหว่างสองจุดยอดใดๆ ดังรูปที่ 11. 2 แสดงตัวอย่างกราฟเส้นทางระหว่างอำเภอต่างๆ ในจังหวัดเลย โดยแต่ละเส้นเชื่อมระหว่างจุดยอดแสดงถึงระยะทางระหว่างอำเภอในหน่วยกิโลเมตร เช่น ระยะทางระหว่างอำเภอเมืองและอำเภอเชียงคานคือ 58 กิโลเมตร เป็นต้น

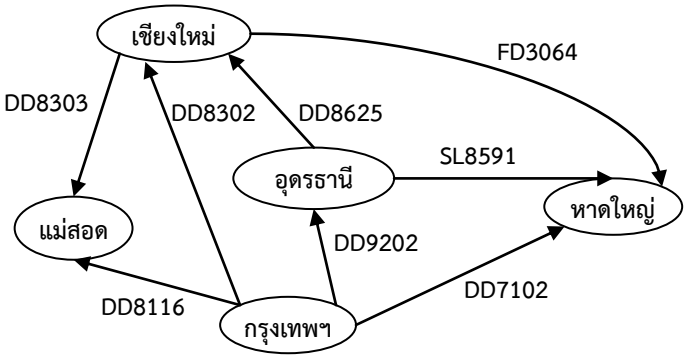


รูปที่ 11.2 ตัวอย่างกราฟแบบไม่มีทิศทาง

2) กราฟแบบมีทิศทาง (Dircted Graph) คือ กราฟที่เส้นเชื่อมมีทิศทางกำกับความสัมพันธ์ระหว่างสองจุดยอดใดๆ เรียกสั้นๆ ว่า ไดกราฟ (Digraph) ตัวอย่างกราฟแบบมีทิศทางในชีวิตประจำวันเช่น เส้นทางในระบบเครือข่ายจะระบุทิศทางการส่งข่าวสารหรือข้อมูลระหว่างอุปกรณ์หรือเครื่องคอมพิวเตอร์ต่างๆ ในระบบเครือข่ายหนึ่งๆ เส้นทางในการบินจะระบุทิศทางการบินระหว่างเมืองต่าง ๆ ของสายการบินหนึ่ง เป็นต้น ดังนั้นความสัมพันธ์ระหว่างสองจุดยอดใดๆ จะมีลักษณะเป็นแบบลำดับ เช่น (U,V) จะหมายความว่า U เป็นจุดยอดเริ่มต้น (Origin) หรือต้นทาง และ V คือจุดยอดปลายทาง (Destination) รูปที่ 11.3 แสดงตัวอย่างกราฟแบบมีทิศทาง โดยจุดยอดแสดงชื่อสนามบิน และเส้นเชื่อมแสดงทิศทางของสายการบินระหว่างสนามบิน เช่น (CNX,BKK) หมายถึงสายการบินที่มีทิศทางการบินจากสนามบินต้นทาง CNX ไปยังสนามบินปลายทาง BKK เป็นต้น

ในกราฟแบบมีทิศทาง หากมีเส้นเชื่อมระหว่างจุดยอดต้นทาง S ไปจุดยอดปลายทาง D จะเรียกเส้นเชื่อมนี้ว่าเส้นเชื่อมออก (Out-edge) ของจุดยอด S แต่จะเป็นเส้นเชื่อมเข้า (In-edge) ของจุดยอด D นั่นคือ ค่าของดีกรีเข้า (In-degree) ของจุดยอดจะเท่ากับจำนวนเส้น

เชื่อมเข้าของจุดยอดนั้น และค่าของดีกรีออก (Out-degree) ของจุดยอดจะเท่ากับจำนวนเส้นเชื่อมออกของจุดยอดนั้น จากรูปที่ 11.3 จุดยอดจังหวัดอุดรธานี มีเส้นเชื่อมเข้าเท่ากับ 1 และเส้นเชื่อมออกเท่ากับ 2 เป็นต้น

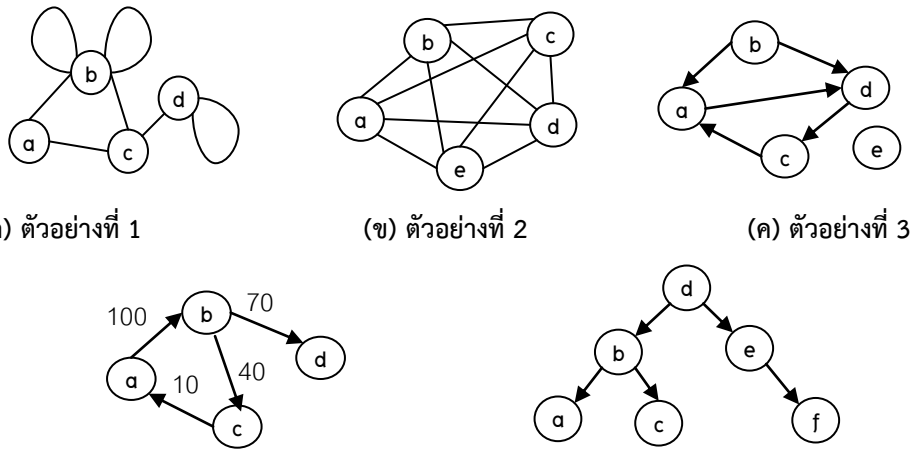


รูปที่ 11.3 ตัวอย่างกราฟแบบมีทิศทาง

(2) ประเภทกราฟอื่นๆ

นอกจากการแบ่งประเภทกราฟตามลักษณะของทิศทางเส้นเชื่อมแล้ว ยังมีการแบ่งกราฟตามลักษณะอื่นๆ ได้อีก รูปที่ 11.4 แสดงรายละเอียดลักษณะของกราฟแต่ละประเภทที่ระบุในตารางที่ 11.1 โดยรูปที่ 11.4 (ก)-(ข) เป็นกราฟแบบมีทิศทาง และรูปที่ 11.4 (ค)-(ง) เป็นลักษณะกราฟแบบไม่มีทิศทางของเส้นเชื่อม ลักษณะกราฟประภทมีดังนี้

1) กราฟมีวง (Cyclic Graph) คือ กราฟที่มีเส้นทางเชิงเดียวแบบมีจุดยอดแรกและจุดยอดสุดท้ายในเส้นทางเป็นจุดเดียวกัน ดังรูปที่ 11. 4 (ก)-(ค) และกราฟไม่มีวง (Acyclic Graph) คือ กราฟที่มีเส้นทางแบบไซเคิลเลย ดังรูปที่ 11.4 (ง) ซึ่งเป็นลักษณะเฉพาะของกราฟแบบต้นไม้
2) กราฟบริบูรณ์ (Complete Graph) คือกราฟที่ทุกคู่ของจุดยอดถูกเชื่อมต่อถึงกัน ดังรูปที่ 11.4 (ข)



(ก) ตัวอย่างที่ 1 **(ข) ตัวอย่างที่ 2** **(ค) ตัวอย่างที่ 3**

(ง) ตัวอย่างที่ 4 **(จ) ตัวอย่างที่ 5**

รูปที่ 11.4 ตัวอย่างกราฟประเภทต่าง ๆ

ตารางที่ 11.1 ลักษณะกราฟประเภทต่าง ๆ

จากรูปที่	ดีกรีของจุดยอด	ตัวอย่างเส้นทาง	ประเภทกราฟ
11.4 (ก)	ดีกรีของ a = 2 ดีกรีของ b = 6 ดีกรีของ c = 3 ดีกรีของ d = 3	เส้นทางความยาว 1 เช่น a-b, a-c , b-c, c-d, b-b, d-d เส้นทางความยาว 2 เช่น a-b-c, a-c-b, a-c-d, a-b-b, c-d-d, d-c-b เส้นทางความยาว 3 เช่น a-b-c-d, a-b-b-c, a-c-d-d, b-b-c-d	- กราฟแบบไม่มีทิศทาง - กราฟมีวงเนื่องจากมีเส้นทางที่เป็นไซเคิลจาก aไปยัง a เช่น a-b-c-a, a-c-b-a - กราฟต่อเนื่อง เนื่องจากทุกๆ คู่ของจุดยอดมีเส้นเชื่อมอยู่
11.4 (ข)	ดีกรีของ a = 4 ดีกรีของ b = 4 ดีกรีของ c = 4 ดีกรีของ d = 4 ดีกรีของ e = 4	เส้นทางความยาว 1 เช่น a-b, a-c, a-d, a-e, b-c, b-d, b-e, c-d, d-e เส้นทางความยาว 2 เช่น a-b-c, a-b-d, a-b-e, b-c-d, c-d-e เส้นทางความยาว 3 เช่น a-b-c-d, a-b-c-e, a-c-d-e, b-c-d-e เส้นทางความยาว 4 เช่น a-b-c-d-e, a-e-b-c-d, a-e-c-d-b, a-b-d-c-a	- กราฟแบบไม่มีทิศทาง - กราฟมีวงเนื่องจากมีเส้นทางที่เป็นไซเคิลจาก a ไปยัง a เช่น a-b-c-a, a-b-e-a, a-d-e-a, a-b-d-c-a, a-b-c-d-e-a - กราฟต่อเนื่อง - กราฟแบบบริบูรณ์
11.4 (ค)	ดีกรีของ a = 3 ดีกรีของ b = 2 ดีกรีของ c = 2 ดีกรีของ d = 3 ดีกรีของ e = 0	เส้นทางความยาว 1 เช่น a-d, b-a, b-d, c-a, d-c เส้นทางความยาว 2 เช่น a-d-c, b-a-d, b-d-c, c-a-d, d-c-a เส้นทางความยาว 3 เช่น a-d-c-a, b-a-d-c, b-d-c-a, c-a-d-c เส้นทางความยาว 4 เช่น b-a-d-c-a, b-d-c-a-d	- กราฟแบบมีทิศทาง - กราฟมีวงเนื่องจากมีเส้นทางที่เป็นไซเคิลจาก a ไปยัง a เช่น a-d-c-a - กราฟไม่เชื่อมต่อ เนื่องจากจุดยอด e ไม่มีเส้นเชื่อมจากจุดยอดอื่นในกราฟ
11.4 (ง)	ดีกรีของ a = 2 ดีกรีของ b = 3 ดีกรีของ c = 2 ดีกรีของ d = 1	เส้นทางความยาว 1 เช่น a-b, b-d, b-c, c-a เส้นทางความยาว 2 เช่น a-b-d,a-b-c, b-c-a, c-a-b เส้นทางความยาว 3 เช่น c-a-b-d	- กราฟแบบมีทิศทาง - กราฟมีวงเนื่องจากมีเส้นทางที่เป็นไซเคิลจาก a ไปยัง a เช่น a-b-c-a - กราฟแบบมีน้ำหนัก เนื่องจากมีค่าน้ำหนักกำกับบนเส้นเชื่อม
11.4 (จ)	ดีกรีของ a = 1 ดีกรีของ b = 3 ดีกรีของ c = 1 ดีกรีของ d = 2 ดีกรีของ e = 2 ดีกรีของ f = 1	เส้นทางความยาว 1 เช่น d-b, b-a, b-c, d-e, e-f เส้นทางความยาว 2 เช่น d-b-a , d-b-c , d-e-f	- กราฟแบบมีทิศทาง - กราฟไม่มีวงเพราะไม่มีเส้นทางที่เป็นไซเคิลและเป็นลักษณะเฉพาะของกราฟที่เป็นแบบโครงสร้างต้นไม้หรือทรี - กราฟต่อเนื่อง

3) กราฟเชื่อมต่อ(Connected Graph) หรือกราฟเชื่อมโยงหรือกราฟต่อเนื่องคือ กราฟไม่ขาดจากกัน หมายถึง แต่ละจุดยอดในกราฟจะมีอย่างน้อยหนึ่งเส้นเชื่อมจากจุดยอดนั้นไปยังจุดยอดอื่นๆ ในกราฟหากมีอย่างน้อยหนึ่งจุดยอดในกราฟที่ไม่มีเส้นเชื่อมไปยังจุดยอดอื่นๆ ในกราฟเลย จะเรียกว่ากราฟไม่ต่อเนื่อง (Disconnected Graph) ดังรูปที่ 11.4 (ค)

จุดยอด e ไม่มีเส้นเชื่อมไปยังจุดยอดใดๆ ในกราฟเลย

4) กราฟถ่วงน้ำหนัก (Weighted Graph) คือ กราฟที่มีการกำหนดค่าให้กับทุกเส้นเชื่อมในกราฟ ซึ่งอาจเป็นค่าใช้จ่ายน้ำหนักความยาว หรืออื่นๆขึ้นกับการใช้งานกราฟ ดังรูปที่ 11.4(ง) แต่ละเส้นเชื่อมจะแสดงความสัมพันธ์ระหว่างคู่จุดยอดในกราฟ เช่น ค่าแสดงความสัมพันธ์ระหว่างจุดยอด a และ b มีค่าน้ำหนักเท่ากับ 100 และค่าแสดงความสัมพันธ์ระหว่างจุดยอด b และ d มีค่าน้ำหนักเท่ากับ 70 เป็นต้น
5) กราฟในลักษณะต้นไม้หรือทรี (Tree) คือ กราฟไม่ขาดจากกัน หมายถึง ทุกคู่จุดยอดในกราฟมีเส้นเชื่อมถึงกัน ดังรูปที่ 11.4 (จ)

11.2 การแวะผ่านกราฟ

การแวะผ่านกราฟ (Graph Traversal) คือ กระบวนการในการท่องไปในกราฟ เพื่อเข้าถึงทุกๆ จุดยอดของกราฟ เพื่อทำการสำรวจ (Explore) หรือเข้าถึงทุกจุดยอดและทุกเส้นเชื่อมในกราฟ วิธีในการแวะผ่านกราฟ มี 2 วิธีดังนี้

1) การค้นหาแบบแนวกว้าง (Breadth-First Search:BFS) เป็นเทคนิคในการแวะผ่านกราฟแบบเข้าถึงจุดยอดหนึ่งๆ จะไปยังทุกจุดยอดที่เป็นจุดยอดใกล้เคียงทั้งหมดของจุดยอดนั้นก่อน แล้วจึงไปยังจุดยอดลูกของจุดยอดนั้นต่อไป ซึ่งจะใช้โครงสร้างข้อมูลคิวช่วยในการจัดเก็บข้อมูลระหว่างการแวะผ่านกราฟดังตัวอย่างอัลกอริทึมต่อไปนี้

```
BFS(G, s) {
  Enqueue(Q, s);
  while (Q not empty) {
    u = Dequeue(Q);
    for each v ∈ u->adj {
        if (v->color == WHITE)
            v->color = GREY;
            Enqueue(Q, v);
    }
    u->color = BLACK;
  }
}
```

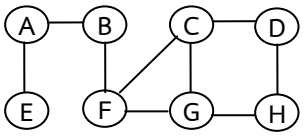
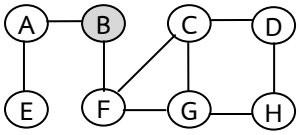
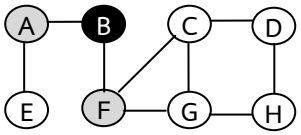
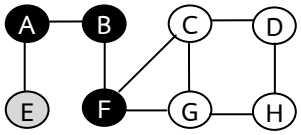
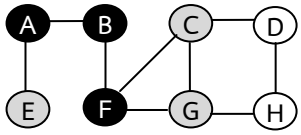
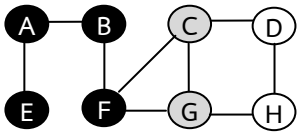
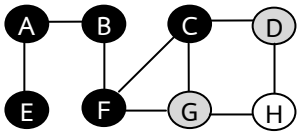
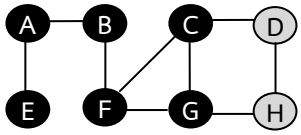
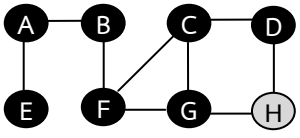
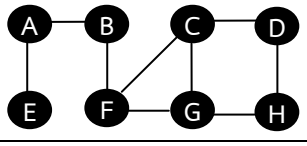
เมื่อนำจุดยอด u ออกจากคิวให้กำหนด
- สถานะของจุดยอด v ซึ่งเป็นจุดยอดใกล้เคียง (ของจุดยอด u) ให้เป็นสีเทาทุกจุดยอด และนำจุดยอดเหล่านั้นใส่ในใส่คิว
- สถานะจุดยอด u เป็นสีดำ

ในระหว่างการท่องกราฟ จะเก็บสถานะของการเข้าถึงแต่ละจุดยอด v ในกราฟไว้ใน v->color โดยแบ่งเป็น 3 สถานะ ดังนี้

- ค่าสีขาว (White)เมื่อจุดยอดนั้นยังไม่เคยถูกค้นพบ ดังนั้น เมื่อเริ่มต้นท่องกราฟ จะกำหนดสถานะทุกจุดยอดในกราฟ เป็นค่าสีขาวนี้
- ค่าสีเทา (Grey)เมื่อจุดยอดนั้นเคยถูกค้นพบแล้วแต่ยังสำรวจไม่เสร็จ เช่น กลุ่มจุดยอดที่เป็นจุดยอดใกล้เคียงของจุดยอดที่มีสถานะเป็นสีขาว เป็นต้น
- ค่าสีดำ (Black)เมื่อจุดยอดนั้นถูกค้นพบและถูกสำรวจเสร็จแล้ว ซึ่งจะเป็นกลุ่มจุดยอดใกล้เคียงกับจุดยอดที่มีสถานะเป็นสีดำหรือสีเทา

การสำรวจแต่ละจุดยอดในกราฟแบบแนวกว้างจะเข้าถึงทีละจุดยอดตามลำดับจุดยอดใกล้เคียงของจุดยอดที่จัดเก็บในคิว ดังตัวอย่างลำดับที่ 2 ในตารางที่ 11.2 จะพบว่าเมื่อจุดยอดใดที่จะถูกจัดเก็บในคิวจะถูกกำหนดสถานะเป็นจุดยอดสีเทา และหากจุดยอดนั้นถูกนำออกจากคิวจะนำจุดยอดใกล้เคียงของจุดยอดนั้นจัดเก็บในคิวแทนพร้อมทั้งกำหนดสถานะจุดยอดใกล้เคียงเหล่านั้นเป็นสีเทา

ตารางที่ 11.2 ลำดับการเปลี่ยนสถานะจุดยอดในกราฟที่มีการเข้าถึงแบบแนวกว้าง

ลำดับ	รูปกราฟ	สถานะคิว	คำอธิบาย
1		ว่าง	เริ่มต้น คิวว่าง และทุกจุดยอดมีสถานะเป็นสีขาว
2		B	นำจุดยอดแรก (ในที่นี้คือจุดยอด B) เข้าคิวและกำหนดสถานะจุดยอดBเป็นสีเทา
3		AF	- นำ B ออกจากคิว และนำจุดยอดเพื่อนบ้านที่มีสถานะเป็นสีขาว (A และ F) ของจุดยอด B ใส่ในคิว และกำหนดสีจุดยอด A และ F เป็นสีเทา - กำหนดสีจุดยอด B เป็นสีดำ
4		F E	- นำ A ออกจากคิว และนำจุดยอดเพื่อนบ้านที่มีสถานะเป็นสีขาว (มีจุดยอดเดียวคือ E) ของจุดยอด A ใส่ในคิว และกำหนดสีจุดยอด E เป็นสีเทา - กำหนดสีจุดยอด A เป็นสีดำ
5		E C G	- นำ F ออกจากคิว และนำจุดยอดเพื่อนบ้านที่มีสถานะเป็นสีขาว (C และ G) ของจุดยอด F ใส่ในคิว และกำหนดสีจุดยอดC และ G เป็นสีเทา - กำหนดสีจุดยอดF เป็นสีดำ
6		C G	- นำ E ออกจากคิว และนำจุดยอดเพื่อนบ้านของจุดยอด E ที่มีสถานะเป็นสีขาวใส่ในคิว ซึ่งในที่นี้ไม่มีจุดยอดที่เป็นสีขาวเหลืออยู่ จึงไม่เพิ่มค่าในคิว - กำหนดสีจุดยอด E เป็นสีดำ
7		G D	- นำ C ออกจากคิว และนำจุดยอดเพื่อนบ้านที่มีสถานะเป็นสีขาว (มีจุดยอดเดียวคือ D) ของจุดยอด C ใส่ในคิว และกำหนดสีจุดยอด D เป็นสีเทา - กำหนดสีจุดยอด C เป็นสีดำ
8		D H	- นำ G ออกจากคิว และนำจุดยอดเพื่อนบ้านที่มีสถานะเป็นสีขาว (มีจุดยอดเดียวคือ H) ของจุดยอด G ใส่ในคิว และกำหนดสีจุดยอดH เป็นสีเทา - กำหนดสีจุดยอด G เป็นสีดำ
9		H	- นำ D ออกจากคิว ซึ่งในที่นี้ไม่มีจุดยอดเพื่อนบ้านของจุดยอด D ที่เป็นสีขาวเหลืออยู่ จึงไม่เพิ่มค่าในคิว - กำหนดสีจุดยอด D เป็นสีดำ
10		ว่าง	- นำ Hออกจากคิว ซึ่งในที่นี้ไม่มีจุดยอดเพื่อนบ้านของจุดยอดH ที่เป็นสีขาวเหลืออยู่ จึงไม่เพิ่มค่าในคิว - กำหนดสีจุดยอดHเป็นสีดำ

ส่วนจุดยอดที่เพิ่งนำออกจากคิวนี้จะกำหนดสถานะเป็นสีดำแทนสีเทา เพื่อระบุสถานะว่าถูกค้นพบและเข้าถึงข้อมูลเสร็จเรียบร้อยแล้ว ดังตัวอย่างลำดับที่ 3-9 ในตารางที่ 11.2 จากนั้นจะวนทำซ้ำกับข้อมูลในคิวต่อไป จนกว่าคิวจะว่างจึงจะเป็นการสิ้นสุดการทำงานของการเข้าถึงจุดยอดในกราฟแบบแนวกว้างครบทุกจุดยอดในกราฟ ดังตัวอย่างลำดับที่ 10 ในตารางที่ 11.2

2) การค้นหาแบบแนวลึก (Depth-First Search : DFS) เป็นการเข้าถึงหรือค้นหาข้อมูลในกราฟตามความลึกของเส้นทางที่เริ่มจากจุดยอดหนึ่งในกราฟไปยังจุดยอดอื่นๆ จนกว่าจะไม่สามารถไปในเส้นทางนั้นได้อีกแล้ว จากนั้นจึงเข้าถึงจุดยอดในกราฟที่เส้นทางอื่นต่อไปจนกว่าจะสามารถเข้าถึงทุกจุดยอดในกราฟ ดังตัวอย่างอัลกอริทึมของการค้นหาแบบแนวลึกต่อไปนี้

```
DFS(G)
    for each u∈V do
        color[u] ← white
    time← 0
    for each u∈V do
        if color[u] = white then
            DFS-VISIT(G, u)
```

เริ่มต้นทุกจุดยอดในกราฟจะถูกกำหนดสถานะเป็นสีขาว และแต่ละจุดยอดจะถูกเข้าถึงด้วยอัลกอริทึม DFS-VISIT()

```
DFS-VISIT(G, u)
    color[u] ← gray
    time ← time +1
    d[u] ← time
    for each v ∈ Adj[u] do
        if color[v] = white then
            DFS-VISIT(G, v)
        color[u] ← black
        time ← time +1
        f[u] ←time
```

แต่ละจุดยอด u จะถูกกำหนดค่า timestamps จำนวน 2 ครั้ง ครั้งแรกเมื่อจุดยอด u ถูกค้นพบจะเก็บค่า timestamps ไว้ใน d[u] และกำหนดสถานะจุดยอดเป็นสีเทา และครั้งที่สอง ค่าที่สองเมื่อจุดยอด u ถูกสำรวจเสร็จแล้ว จะเก็บค่า timestamps ไว้ใน f[u] และกำหนดสถานะจุดยอด u นั้นเป็นสีดำ

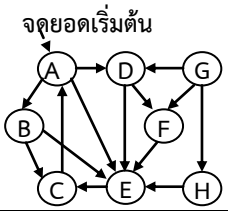
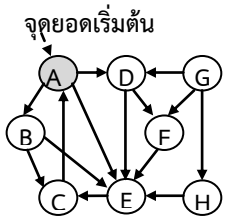
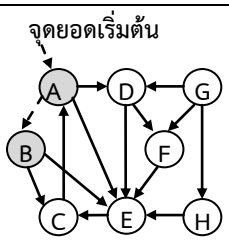
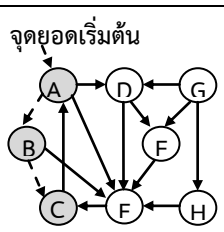
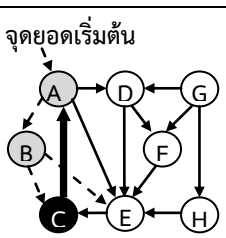
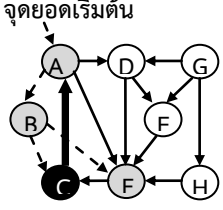
จากอัลกอริทึมข้างต้นการเข้าถึงจุดยอดในกราฟแบบแนวลึกจะใช้โครงสร้างข้อมูลสแตกเข้ามาช่วยในการจัดเก็บข้อมูลระหว่างการแวะผ่านจุดยอด หรือใช้วิธีการแบบ recursive มาช่วย โดยมีการกำหนดสถานะของจุดยอดเป็น 3 สี เช่นเดียวกับการค้นหาแบบแนวกว้าง คือ สีขาว หมายถึง จุดยอดยังไม่ถูกค้นพบ สีเทา หมายถึง จุดยอดถูกค้นพบแล้ว แต่สีดำ หมายถึง ถูกสำรวจหรือเข้าถึงข้อมูลเสร็จแล้วซึ่งสามารถแสดงตัวอย่างการเข้าถึงทุกจุดยอดในกราฟได้ ดังตารางที่ 11.3 มีรายละเอียดดังนี้

1) เริ่มต้นทุกจุดยอดจะถูกกำหดนสถานะสีขาว
2) เมื่อจุดยอด u ถูกเข้าถึง
   2.1) กำหนดสถานะจุดยอดเป็นสีเทา (color[u]← gray) จากนั้นให้เพิ่มค่า time เป็น 1 และกำหนดให้ d[u] ← time
   2.2) จากจุดยอด u จะค้นหาว่าจุดยอดใกล้เคียงของจุดยอด u ใดมีสถานะเป็นสีขาว เพื่อจะถูกเข้าถึงจุดยอดนั้นในลำดับถัดไปด้วยการเรียกซ้ำ (Recursive)

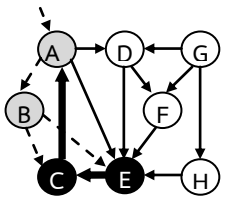
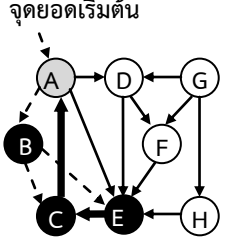
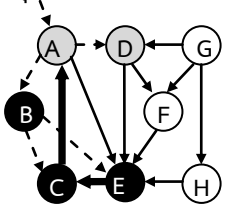
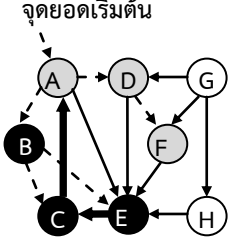
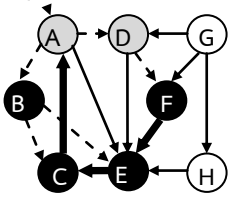
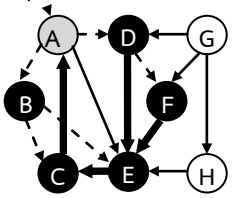
3) หากจุดยอดใกล้เคียงของจุดยอด u ไม่มีจุดยอดใดมีสถานะเป็นสีขาวอีกแล้วจะกำหนดสถานะจุดยอด u เป็นสีดำ (color[u]← black) จากนั้นให้เพิ่มค่า time เป็น 1 และกำหนดf[u]←time เพื่อบอกสถานะว่าสิ้นสุดการเข้าถึงข้อมูลทั้งหมดในจุดยอดu

4) วนทำซ้ำข้อ 2 กับจุดยอดอื่นๆ ที่มีสถานะเป็นสีขาวจนครบทุกจุดยอดในกราฟ

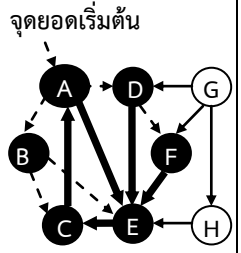
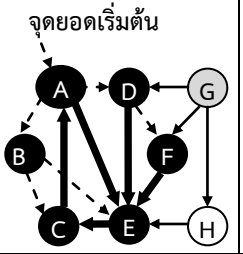
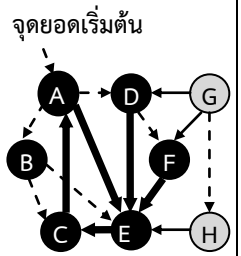
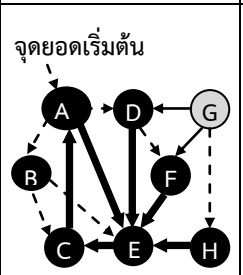
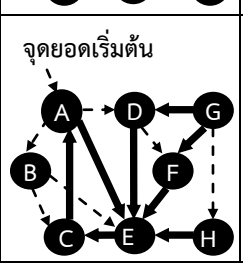
ตารางที่ 11.3 ลำดับการเปลี่ยนสถานะจุดยอดในกราฟที่มีการเข้าถึงแบบแนวลึก

ค่า time	รูปกราฟ	จุดยอด u	ค่า d[u],f[u]	คำอธิบาย
0				- สถานะเริ่มต้นทุกจุดยอดมีสถานะเป็นสีขาว
1		A	d[u]=1	- กำหนดสถานะจุดยอด A เป็นสีเทา - พิจารณาจุดยอดใกล้เคียงของ A พบว่ามีจุดยอด B ที่มีสถานะเป็นสีขาว จึงเข้าถึงจุดยอด B ต่อไป
2		B	d[u]=2	- กำหนดสถานะจุดยอด B เป็นสีเทา - พิจารณาจุดยอดใกล้เคียงของ B พบว่ามีจุดยอด C ที่มีสถานะเป็นสีขาว จึงเข้าถึงจุดยอด C ต่อไป
3		C	d[u]=3	- กำหนดสถานะจุดยอด C เป็นสีเทา - พิจารณาจุดยอดใกล้เคียงของ C พบว่าไม่มีจุดยอดใกล้เคียงใดมีสถานะสีขาว
4		C	d[u]=3, f[u]=4	- กำหนดสถานะจุดยอด C เป็นสีดำ เพราะเสร็จสิ้นการเข้าถึงจุดยอดใกล้เคียงของ C - ย้อนกลับไปพิจารณาจุดยอดใกล้เคียงของจุดยอด B พบว่ามีจุดยอด E ที่มีสถานะเป็นสีขาว จึงเข้าถึงจุดยอด E ต่อไป
5		E	d[u]=5	- กำหนดสถานะจุดยอด E เป็นสีเทา - พิจารณาจุดยอดใกล้เคียง E พบว่าไม่มีจุดยอดใกล้เคียงใดมีสถานะสีขาว

ตารางที่ 11.3 ลำดับการเปลี่ยนสถานะจุดยอดในกราฟที่มีการเข้าถึงแบบแนวลึก

ค่า time	รูปกราฟ	จุดยอด u	ค่า d[u],f[u]	คำอธิบาย
6	จุดยอดเริ่มต้น 	E	d[u]=5, f[u]=6	- กำหนดสถานะจุดยอด E เป็นสีดำเพราะเสร็จสิ้นการเข้าถึงจุดยอดใกล้เคียงของ E - ย้อนกลับไปพิจารณาจุดยอดใกล้เคียงของจุดยอด B พบว่าไม่มีจุดยอดใดที่มีสถานะเป็นสีขาว
7	จุดยอดเริ่มต้น 	B	d[u]=2, f[u]=7	- กำหนดสถานะจุดยอด B เป็นสีดำเนื่องจากเสร็จสิ้นการเข้าถึงจุดยอดใกล้เคียงของ B - ย้อนกลับไปพิจารณาจุดยอดใกล้เคียงของ A พบว่ามีจุดยอด D ที่มีสถานะเป็นสีขาว จึงเข้าถึงจุดยอด D ต่อไป
8	จุดยอดเริ่มต้น 	D	d[u]=8	- กำหนดสถานะจุดยอด D เป็นสีเทา - พิจารณาจุดยอดใกล้เคียงของ D พบว่ามีจุดยอด F ที่มีสถานะเป็นสีขาว จึงเข้าถึงจุดยอด F ต่อไป
9	จุดยอดเริ่มต้น 	F	d[u]=9	- กำหนดสถานะจุดยอด F เป็นสีเทา - พิจารณาจุดยอดใกล้เคียงของ F พบว่าไม่มีจุดยอดใกล้เคียงใดมีสถานะสีขาว
10	จุดยอดเริ่มต้น 	F	d[u]=9 f[u]=10	- กำหนดสถานะจุดยอด F เป็นสีดำเนื่องจากเสร็จสิ้นการเข้าถึงจุดยอดใกล้เคียงของ F - ย้อนกลับไปพิจารณาจุดยอดใกล้เคียงของ D พบว่าไม่มีจุดยอดใดที่มีสถานะเป็นสีขาว
11	จุดยอดเริ่มต้น 	D	d[u]=8 f[u]=11	- กำหนดสถานะจุดยอด D เป็นสีดำเนื่องจากเสร็จสิ้นการเข้าถึงจุดยอดใกล้เคียงของ D - ย้อนกลับไปพิจารณาจุดยอดใกล้เคียงของ A พบว่าไม่มีจุดยอดใดที่มีสถานะเป็นสีขาว

ตารางที่ 11.3 ลำดับการเปลี่ยนสถานะจุดยอดในกราฟที่มีการเข้าถึงแบบแนวลึก

ค่า time	รูปกราฟ	จุดยอด u	ค่า d[u],f[u]	คำอธิบาย
12		A	d[u]=1 f[u]=12	- กำหนดสถานะจุดยอด A เป็นสีดำ เนื่องจากเสร็จสิ้นการเข้าถึงจุดยอดใกล้เคียงของ A - พิจารณาจุดยอดอื่น ๆ ในกราฟที่มีสถานะสีขาว ในที่นี้คือจุดยอด G
13		G	d[u]=13	- กำหนดสถานะจุดยอด G เป็นสีเทา - พิจารณาจุดยอดใกล้เคียงของ G พบว่ามีจุดยอด H ที่มีสถานะเป็นสีขาว จึงเข้าถึงจุดยอด H ต่อไป
14		H	d[u]=14	- กำหนดสถานะจุดยอด H เป็นสีเทา - พิจารณาจุดยอดใกล้เคียงของ H พบว่าไม่มีจุดยอดใกล้เคียงใดมีสถานะสีขาว
15		H	d[u]=14 f[u]=15	- กำหนดสถานะจุดยอด H เป็นสีดำ เนื่องจากเสร็จสิ้นการเข้าถึงจุดยอดใกล้เคียงของ H - ย้อนกลับไปพิจารณาจุดยอดใกล้เคียงของ G พบว่าไม่มีจุดยอดใดที่มีสถานะเป็นสีขาว
16		G	d[u]=13 f[u]=16	- กำหนดสถานะจุดยอด G เป็นสีดำ เนื่องจากเสร็จสิ้นการเข้าถึงจุดยอดใกล้เคียงของ G - สิ้นสุดการเข้าถึงทุกจุดยอดในกราฟ

11.3 การนำกราฟไปใช้

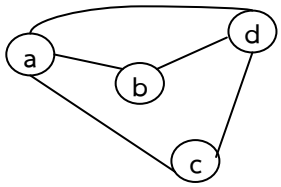
การนำโครงสร้างข้อมูลกราฟไปใช้ในการจัดเก็บข้อมูลจะขึ้นอยู่กับลักษณะการจัดเก็บข้อมูล ซึ่งจะมีลักษณะการจัดเก็บอยู่ 2 รูปแบบ มีรายละเอียดดังนี้

11.3.1 การจัดเก็บโครงสร้างกราฟด้วยแอดจาเซนซีเมทริกซ์

วิธีการอย่างง่ายในการจัดเก็บโครงสร้างกราฟ คือ อาร์เรย์สองมิติที่เก็บความสัมพันธ์ระหว่างจุดยอดใกล้เคียงในกราฟ เรียกว่า แอดจาเซนซีเมทริกซ์ (Adjacency Matrix) โดยหากกราฟ

ประกอบด้วยจุดยอด v_1, v_2, v_3, ... , v_n จะสามารถแสดงกราฟในรูปของแอดจาเซนซีเมทริกซ์ X ขนาด n x n โดย x_{ij}= 1 ในกรณีมีเส้นเชื่อมระหว่างจุดยอด v_i และ v_j และ x_{ij}= 0 ในกรณีไม่มีเส้นเชื่อมระหว่างจุดยอด v_i และ v_j และ i, j ≤ n

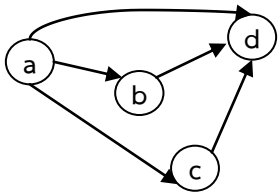
รูปที่ 11.5 (ข) แสดงการจัดเก็บโครงสร้างกราฟแบบไม่มีทิศทางของกราฟในรูปที่ 11. 5 (ก) โดยแถวที่ 1 ของอาร์เรย์ X แสดงการเชื่อมโยงจากจุดยอด a ไปยังจุดยอดอื่น ๆ ที่เป็นจุดยอดใกล้เคียง เช่น จุดยอด a เชื่อมโยงไปยังจุดยอด b c และ d ดังนั้น ค่า x_{12} x_{13} และ x_{14} จะมีค่าเป็น 1 ส่วนแถวที่ 2-4 แสดงการเชื่อมโยงจากจุดยอด b c หรือ d ไปยังจุดยอดอื่นๆ ตามลำดับ ซึ่งการแทนกราฟแบบไม่มีทิศทาง ลักษณะของอาร์เรย์สองมิติที่ได้จะมีลักษณะสมมาตร (Symmetric) เรียกว่าเมทริกซ์ทแยงมุม (Diagonal Matrix) นั่นคือ ค่าของ x_{ij} = x_{ji} ซึ่งเป็นค่าที่แสดงความสัมพันธ์ระหว่างจุดยอดi และ จุดยอด j ว่ามีเส้นชื่อมโยงถึงกัน ดังนั้น จำนวนช่อง x_{ij} ที่มีค่าเท่ากับ 1 จะเท่ากับสองเท่าของจำนวนเส้นเชื่อมในกราฟ แต่หากเป็นกราฟแบบมีทิศทาง จำนวนช่อง x_{ij} ที่มีค่าเท่ากับ 1 จะเท่ากับจำนวนเส้นเชื่อมในกราฟ เช่น x_{14} และ x_{41} จะเก็บค่า 1 หมายถึงมีเส้นเชื่อมโยงระหว่างจุดยอด a และจุดยอด d นั่นเอง



(ก) ตัวอย่างกราฟแบบไม่มีทิศทาง

X =	a	b	c	d
a	0	1	1	1
b	1	0	0	1
c	1	0	0	1
d	1	1	1	0

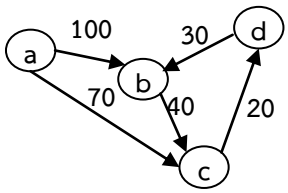
(ข) การจัดเก็บโครงสร้างกราฟแบบไม่มีทิศทาง



(ค) ตัวอย่างกราฟแบบมีทิศทาง

X =	a	b	c	d
a	0	1	1	1
b	0	0	0	1
c	0	0	0	1
d	0	0	0	0

(ง) การจัดเก็บโครงสร้างกราฟแบบมีทิศทาง



(จ) ตัวอย่างกราฟถ่วงน้ำหนัก

X =	a	b	c	d
a	0	100	70	0
b	0	0	40	0
c	0	0	0	20
d	0	30	0	0

(ฉ) การจัดเก็บโครงสร้างกราฟถ่วงน้ำหนัก

รูปที่ 11.5 ตัวอย่างการจัดเก็บโครงสร้างกราฟด้วยแอดจาเซนซีเมทริกซ์

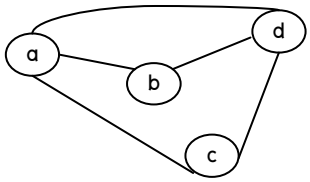
สำหรับตัวอย่างกราฟแบบมีทิศทางในรูปที่ 11. 5 (ค) สามารถแสดงการจัดเก็บโครงสร้างกราฟในรูปแบบอาร์เรย์ได้ดังรูปที่ 11. 5 (ง) ซึ่งจะพบว่าค่า x_{14}เป็น 1 แต่ x_{41} มีค่าเป็น 0 เพราะเส้นเชื่อมโยงระหว่างจุดยอด a และ d มีทิศทางจากจุดยอดa ไปยังจุดยอด d เพียงทิศทางเดียวเท่านั้น ส่วนตัวอย่างกราฟถ่วงน้ำหนักในรูปที่ 11.5 (จ) เป็นลักษณะกราฟแบบมีทิศทางที่มีการกำหนดค่าให้กับทุกเส้นเชื่อม ดังนั้นค่าที่จัดเก็บในอาร์เรย์สองมิติจะเป็นค่าน้ำหนักของแต่ละเส้นเชื่อมที่แสดง

ความสัมพันธ์ระหว่างคู่จุดยอดในกราฟ เช่น เส้นเชื่อมจากจุดยอด a ไป b มีค่าน้ำหนักแสดงความสัมพันธ์ระหว่างจุดยอดa และ b เท่ากับ 100 ดังนั้น x_{12} เป็น 100 ส่วนเส้นเชื่อมจากจุดยอด d ไป b มีค่าน้ำหนักแสดงความสัมพันธ์ระหว่างจุดยอดb และ d เท่ากับ 30 ดังนั้น x_{42} เป็น 30 เป็นต้น

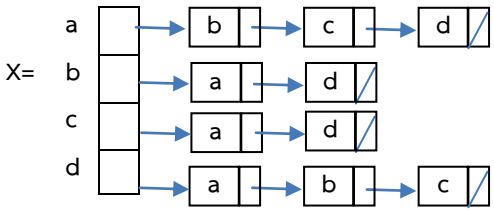
11.3.2 การจัดเก็บโครงสร้างกราฟด้วยแอดจาเซนซีลิสต์

อย่างไรก็ตามบางครั้งการจัดเก็บข้อมูลกราฟขนาด n ในรูปแบบ แอดจาเซนซีเมทริกซ์ ด้วยอาร์เรย์สองมิติขนาด n^2 ในกรณีที่ n มีขนาดใหญ่มากๆ จะทำให้ใช้เนื้อที่หน่วยความจำมากเกินไป วิธีหนึ่งที่ช่วยลดการใช้งานพื้นที่หน่วยความจำ คือ การแทนกราฟด้วยอาร์เรย์ของลิงค์ลิสต์ เรียกว่า แอดจาเซนซีลิสต์ (Adjacency Lists) โดยอาร์เรย์แต่ละช่องจะหมายถึงลำดับจุดยอดในกราฟ โดยแต่ละช่องจะเชื่อมโยงไปยังลิงค์ลิสต์ของจุดยอดเพื่อนบ้านของจุดยอดนั้น ดังตัวอย่างกราฟในรูปที่ 11.6 (ก) สามารถแสดงการจัดเก็บโครงสร้างกราฟด้วยแอดจาเซนซีลิสต์ได้ โดยใช้อาร์เรย์หนึ่งมิติ X เก็บค่าตัวเชื่อมโยงไปยังรายการจุดยอดใกล้เคียงของจุดยอดต่างๆ ดังรูปที่ 11.6 (ข) ตัวอย่างเช่น

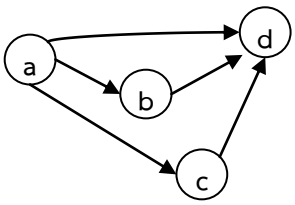
1) จุดยอด a มีจุดยอด b c และ d เป็นจุดยอดใกล้เคียงกัน ดังนั้น X[1] จะเก็บค่าตัวเชื่อมโยงไปยังลิงค์ลิสต์ของจุดยอด b c และ d
2) จุดยอด b มีจุดยอดa และ d เป็นจุดยอดใกล้เคียง ดังนั้น X[2] จะเก็บค่าตัวเชื่อมโยงไปยังลิงค์ลิสต์ของจุดยอด b c และ d เป็นต้น



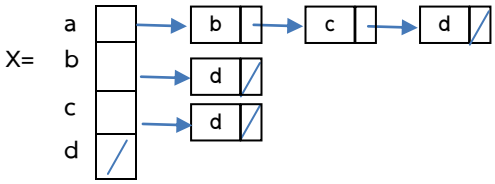
(ก) ตัวอย่างกราฟแบบไม่มีทิศทาง



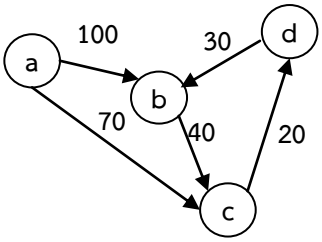
(ข) การจัดเก็บโครงสร้างกราฟแบบไม่มีทิศทาง



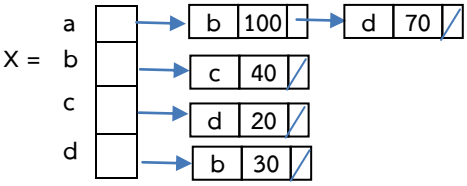
(ค) ตัวอย่างกราฟแบบมีทิศทาง



(ง) การจัดเก็บโครงสร้างกราฟแบบมีทิศทาง



(จ) ตัวอย่างกราฟถ่วงน้ำหนัก



(ฉ) การจัดเก็บโครงสร้างกราฟถ่วงน้ำหนัก

รูปที่ 11.6 ตัวอย่างการจัดเก็บโครงสร้างกราฟด้วยแอดจาเซนซีลิสต์

รูปที่ 11.6 (ค)-(ง) แสดงการจัดเก็บโครงสร้างกราฟแบบมีทิศทางด้วยแอดจาเซนซีลิสต์ ซึ่งจะเก็บข้อมูลในลักษณะเดียวกับตัวอย่างในรูปที่ 11.6 (ก)-(ข) ยกเว้นกรณีจุดยอด d ไม่มีลิงค์เชื่อมโยงในทิศทางเข้าจึงทำให้ X[4] มีค่าเป็น Null รูปที่ 11.6 (จ) แสดงตัวอย่างกราฟถ่วงน้ำหนัก จะจัดเก็บค่าน้ำหนักของแต่ละเส้นเชื่อมที่แสดงความสัมพันธ์ระหว่างคู่จุดยอดในกราฟด้วย เช่น เส้นเชื่อมจากจุดยอด a ไป b มีค่าน้ำหนักแสดงความสัมพันธ์ระหว่างจุดยอด a และ b เท่ากับ 100 ดังนั้น X[1] จะเก็บตัวเชื่อมโยงไปยังลิงค์ลิสต์ของรายการจุดยอดใกล้เคียงของจุดยอด a ซึ่งจะพบว่าในรายการนี้มีการเก็บค่าจุดยอดของจุดยอด bและค่าน้ำหนัก 1 00 เพื่อแสดงความสัมพันธ์จากจุดยอด aไปยังจุดยอด b มีค่าน้ำหนักของเส้นเชื่อมเป็น 100 เป็นต้น

11.3.3 ตัวอย่างคำสั่งภาษาซีสำหรับการจัดเก็บโครงสร้างกราฟด้วยแอดจาเซนซีลิสต์

การออกแบบการจัดเก็บข้อมูลโครงสร้างกราฟ ในหัวข้อนี้เป็นตัวอย่างภาษาซีสำหรับจัดเก็บข้อมูลกราฟในลักษณะของแอดจาเซนซีลิสต์

ตัวอย่างการออกแบบโครงสร้างข้อมูลกราฟ ดังนี้

1) กำหนดโครงสร้างชื่อ AdjList สำหรับจัดเก็บลิงค์ลิสต์ของข้อมูลรายการจุดยอดใกล้เคียง (Adjacency List) ของจุดยอดใดๆ ในกราฟ ดังนี้

```
struct AdjList{
    struct AdjListNode *head;
};
```

แต่ละโหนดใน AdjList จะเก็บข้อมูลในแต่ละจุดยอดใกล้เคียง (Adjacency List) ประกอบด้วยหมายเลขโหนด (vertex_no) และตัวเชื่อมโยงไปยังโหนดถัดไป (next) ในลิสต์ โครงสร้างของโหนดมีรายละเอียดดังนี้

```
struct AdjListNode {
    int vertex_no;
    struct AdjListNode *next;
};
```

2) กำหนดโครงสร้างชื่อ Graph สำหรับจัดข้อมูลกราฟประกอบด้วย อาร์เรย์หนึ่งมิติชื่อ arrAdj แต่ละช่องอาร์เรย์จะเก็บตัวเชื่อมโยงไปยังลิงค์ลิสต์ของรายการจุดยอดใกล้เคียง (Adjacency List) ของจุดยอดนั้นๆ ดังนี้

```
struct Graph {
    int noofvertex;
    struct AdjList* arrAdj;
};
```

ค่าที่เก็บใน arrAdj[1] คือ ตัวชี้ไปยังลิงค์สิสต์ของรายการจุดยอดใกล้เคียงของจุดยอดหมายเลข 1 ค่าที่เก็บใน arrAdj[2] คือ ตัวชี้ไปยังลิงค์สิสต์ของรายการจุดยอดใกล้เคียงของจุดยอดหมายเลข 2 ตามลำดับไปเรื่อยๆ หากตัวแปร noofvertex เก็บจำนวนจุดยอดในกราฟ ดังนั้นค่าที่เก็บใน arrAdj[noofvertex] จะเป็นค่าตัวชี้ไปยังลิงค์สิสต์ของรายการจุดยอดใกล้เคียงของจุดยอดหมายเลข noofvertex นั่นเอง

ตัวอย่างภาษาซีของเมท็อดต่างๆ ของโครงสร้างข้อมูลกราฟ

1) เมท็อดสำหรับสร้างโหนดสำหรับเก็บข้อมูลของจุดยอด เพื่อเตรียมสำหรับเก็บใน AdjListNode ดังตัวอย่างคำสั่งภาษาซีต่อไปนี้

```
struct AdjListNode* newAdjListNode(int v) {
    struct AdjListNode* newNode =(struct AdjListNode*) malloc(sizeof(struct
    AdjListNode));
    newNode -> vertex_no = v;
    newNode -> next = NULL;
    return newNode;
}
```

2) เมท็อดสำหรับการสร้างโครงสร้างข้อมูลกราฟเพื่อจัดเก็บข้อมูลจำนวน n จุดยอด ซึ่งจะมีการจองพื้นที่แบบไดนามิก (Dynamic) สำหรับอาร์เรย์หนึ่งมิติขนาด n ชื่อ arrAdj สำหรับจัดเก็บตัวเชื่อมโยงไปยังลิงค์ลิสต์ของรายการจุดยอดใกล้เคียง เนื่องจากในสถานะเริ่มต้นจะได้กราฟว่าง ดังนั้นจะกำหนดแต่ละ arr Adj[i]=NULL เมื่อ i=1...n เพราะแต่ละค่ายังเป็นลิสต์ว่าง ดังตัวอย่างคำสั่งภาษาซีต่อไปนี้

```
struct Graph* createGraph(int n) {
    int i;
    struct Graph* graph =  (struct Graph*)malloc(sizeof(struct Graph));
    graph -> noofvertex = n;
    graph->arrAdj = (struct AdjList*)malloc(n * sizeof(struct AdjList));
    for (i = 0; i < n; i++)
        graph->arrAdj[i].head = NULL;
    return graph;
}
```

3) เมท็อดสำหรับจัดเก็บข้อมูลเส้นเชื่อมโยงระหว่างคู่จุดยอด src และ dest โดยสร้างโหนดข้อมูล AdjListNodเพื่อเตรียมสำหรับเพิ่มโครงสร้างกราฟ โดยจะแทรกไว้ที่ด้านหน้าของลิสต์ทั้งในลิงค์ลิสต์ graph ->arrAdj[scr].head; และgraph ->arrAdj[dest].head; เนื่องจากเป็นกราฟแบบไม่มีทิศทาง ดังนั้นหากมีเส้นเชื่อมระหว่างจุดยอด src และ dest จะหมายถึง จุดยอด src จะมีจุดยอด dest เป็นจุดยอดใกล้เคียง และจุดยอด dest จะมีจุดยอด src เป็นจุดยอดใกล้เคียงด้วยเช่นกัน ดังตัวอย่างคำสั่งภาษาซีต่อไปนี้

```
void addEdge(struct Graph *graph, int src, int dest) {
    struct AdjListNode* newNode =  newAdjListNode(dest);
    newNode -> next = graph ->arrAdj[src].head;
    graph->arrAdj[src].head = newNode;
    newNode = newAdjListNode(src);
    newNode -> next = graph ->arrAdj[dest].head;
    graph->arrAdj[dest].head = newNode;
}
```

4) เมท็อดสำหรับแสดงข้อมูลในกราฟ โดยจะแสดงรายการของจุดยอดใกล้เคียงของแต่ละจุดยอด v ที่จัดเก็บในลิงค์ลิสต์ graph ->arrAdj[v].head ดังตัวอย่างคำสั่งภาษาซีต่อไปนี้

```
void printGraph(struct Graph* graph){
   int v;
   for (v = 0; v < graph -> noofvertex;  v++) {
        struct AdjListNode* p = graph ->arrAdj[v].head;
        cout<<"\n Adjacency list of vertex "<< v << "\n" ;
        while (p != NULL) {
             cout<< "-> " << p -> vertex_no;
             p = p->next;
        }
        cout<<"\n";
   }
}
```

5) เมท็อดสำหรับเข้าถึงและแสดงค่าทุกจุดยอดในกราฟโดยใช้วิธีการเข้าถึงแบบแนวลึกดังตัวอย่างคำสั่งภาษาซีต่อไปนี้

```
void dfs(struct Graph *graph, int d[]){
     int i;
     int n = graph -> noofvertex;
     t = 0;
     for (i = 0; i < n; i++)
          if(d[i] == 0)dfsvisit(graph, d, i );
}

void dfsvisit(struct Graph *graph, int d[], int v){
  int i; int n = graph -> noofvertex;
  t = t +1;
  d[v] = t;      cout<< " "<<  v ;
  for (i = 0; i < n; i++){
       if (isAdjecent(graph, v, i) && d[i] == 0)
          dfsvisit(graph, d, i);
  }
}
```

พิจารณาแต่ละจุดยอดในกราฟ

ค้นหาจุดยอดใกล้เคียงของจุดยอด v ที่มีสถานะยังไม่เคยถูกค้นพบ (สถานะสีขาว คือ d[i] = 0) แล้วเข้าถึงจุดยอดนั้นแบบแนวลึกโดยการเรียกใช้เมท็อด dfsvisit() แบบเรียกซ้ำ

6) เมท็อดสำหรับตรวจสอบว่าจุดยอด dest เป็นจุดยอดใกล้เคียงของจุดยอด src หรือไม่ หากใช่ให้คืนค่า 1 หากไม่ใช่ให้คืนค่า 0 ดังตัวอย่างคำสั่งภาษาซีต่อไปนี้

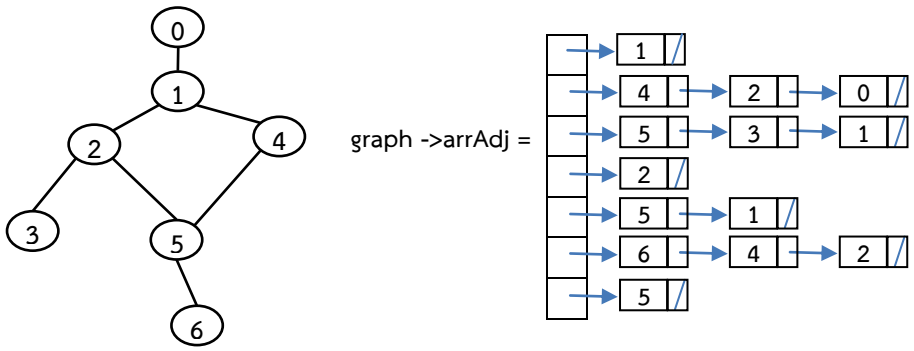
```
int isAdjecent(struct Graph *graph, int src, int dest){
     struct AdjListNode* p = graph->arrAdj[src].head;
     while(p != NULL) {
          if(p->vertex_no == dest) return 1;
          p = p->next;
     }
     return 0;
}
```

ตัวอย่างคำสั่งภาษาซีในการจัดการข้อมูลโครงสร้างกราฟที่จัดเก็บในลักษณะแอดจาเซนซีลิสต์ สามารถแสดงตัวอย่างคำสั่งภาษาซีในการสร้างกราฟตัวอย่างในรูปที่ 11.7 ได้ดังนี้

```
int main() {
    int V = 7;
    int d[7] = {0};
    struct Graph* graph = createGraph(V);
    addEdge(graph, 0, 1);
    addEdge(graph, 1, 2);
    addEdge(graph, 1, 4);
    addEdge(graph, 2, 3);
    addEdge(graph, 2, 5);
    addEdge(graph, 4, 5);
    addEdge(graph, 5, 6);
    printGraph(graph);
    dfs(graph,d);
    return 0;
}
```

จุดยอดทั้ง 7 จุดในกราฟจะถูกกำหนดสถานะเริ่มต้นเป็นสีขาว

คำสั่ง addEdge() ใช้สำหรับเพิ่มแต่ละเส้นเชื่อมระหว่างคู่จุดยอดใดๆ ในกราฟที่ประกอบด้วยจุดยอดหมายเลข 0 ถึง 6 ดังรูปที่ 11.7 (ก) ซึ่งจะมีการจัดเก็บข้อมูลใน ลักษณะของแอดจาเซนซีลิสต์ ดังรูปที่ 11.7 (ข) โดย graph->arrAdj[i] จะเก็บรายการจุดยอดใกล้เคียงของแต่ละจุดยอดหมายเลข i เช่น ลิงค์ลิสต์ของ graph ->arrAdj[1] แสดงรายการจุดยอดใกล้เคียงของ จุดยอดหมายเลข 1 ได้แก่ 0 2 และ 4 แต่เนื่องจากการแทรกโหนดในลิงค์ลิสต์เป็นแบบแทรกด้านหน้าลิสต์จึงเก็บเป็นลำดับ 4 2 และ 0 แทนดังรูปที่ 11.7



(ก) ตัวอย่างกราฟถ่วงน้ำหนัก **(ข) การจัดเก็บโครงสร้างกราฟ**

รูปที่ 11.7 ตัวอย่างการจัดเก็บโครงสร้างกราฟด้วยแอดจาเซนซีลิสต์จากตัวอย่างภาษาซี

สำหรับคำสั่ง printGraph(graph); จะแสดงผลลัพธ์จากข้อมูลที่จัดเก็บใน graph ->arrAdj จะแสดงผลบนจอภาพดังนี้

```
Adjacency list of vertex0
->1
 Adjacency list of vertex1
->4-> 2-> 0
 Adjacency list of vertex2
```

```
-> 5-> 3-> 1
 Adjacency list of vertex3
-> 2
 Adjacency list of vertex4
-> 5-> 1
 Adjacency list of vertex5
-> 6-> 4-> 2
 Adjacency list of vertex6
-> 5
```

ส่วนคำสั่ง dfs(graph,d); จะเข้าถึงทุกจุดยอดในกราฟแบบแนวลึก ดังนั้นจะแสดงหมายเลขจุดยอดในกราฟตามลำดับ ดังนี้

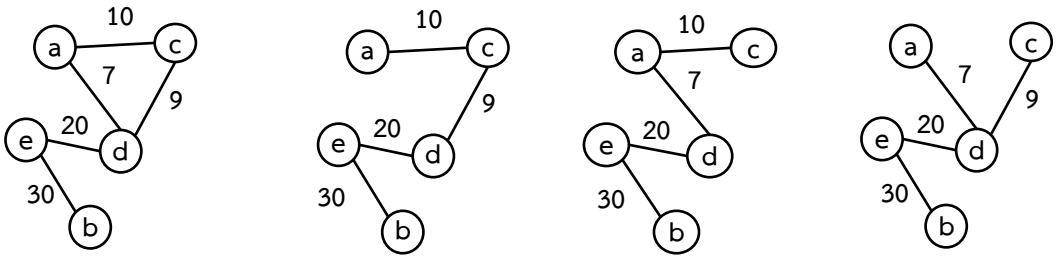
0 1 2 3 5 4 6

11.4 การประยุกต์ใช้งานกราฟ

โครงสร้างข้อมูลกราฟถูกนำไปประยุกต์ใช้ในการแก้ไขปัญหาต่างๆ โดยเฉพาะปัญหาด้านเส้นทาง เช่น การหาทรีแบบทอดข้ามที่เล็กที่สุด การหาเส้นทางที่สั้นที่สุด เป็นต้น

11.4.4 ทรีแบบทอดข้ามที่เล็กที่สุด

ทรีแบบทอดข้าม (Spanning Tree) เป็นทรีที่ประกอบด้วยทุกจุดยอดในกราฟ โดยโครงสร้างกราฟหนึ่งๆ สามารถมีทรีแบบทอดข้ามได้มากกว่า 1 รูปแบบ ดังตัวอย่างในรูปที่ 11. 8 (ก) สามารถแสดงทรีแบบทอดข้ามได้ 3 รูปแบบดังรูปที่ 11.8 (ข)-(ง)



(ก) ตัวอย่างกราฟ (ข) ทรีแบบทอดข้ามรูปที่ 1 (ค) ทรีแบบทอดข้ามรูปที่ 2 (ง) ทรีแบบทอดข้ามรูปที่ 3

รูปที่ 11.8 ตัวอย่างทรีแบบทอดข้าม

ทรีแบบทอดข้ามที่เล็กที่สุด (Minimum Spanning Tree) คือ ทรีแบบทอดข้ามที่มีค่าผลรวมของน้ำหนักบนเส้นเชื่อมน้อยที่สุด จากตัวอย่างในรูปที่ 11. 8 (ข)-(ง) เป็นทรีแบบทอดข้ามที่มีผลรวมของน้ำหนักดังนี้

1) รูปที่ 11.8 (ข) เป็นทรีแบบทอดข้ามที่มีผลรวมของค่าน้ำหนักบนเส้นเชื่อมเท่ากับ
   10+9+20+30 = 69
2) รูปที่ 11.8 (ค) เป็นทรีแบบทอดข้ามที่มีผลรวมของค่าน้ำหนักบนเส้นเชื่อมเท่ากับ
   10+7+20+30 = 67
3) รูปที่ 11.8 (ง) เป็นทรีแบบทอดข้ามที่มีผลรวมของค่าน้ำหนักบนเส้นเชื่อมเท่ากับ
   9+7+20+30 = 66

จากตัวอย่างกราฟในรูปที่ 11.8 (ก) จะได้ทรีแบบทอดข้ามที่เล็กที่สุดดังรูปที่ 11.8 (ง) ซึ่งจะเห็นว่ากราฟหนึ่งๆ จะมีทรีแบบทอดข้ามได้มากกว่าหนึ่งรูปแบบ และการหาทรีแบบทอดข้ามที่มีค่า

น้ำหนักรวมที่น้อยที่สุด ค่อนข้างยุ่งยากเพราะต้องทำการเลือกทีละเส้นเชื่อม จึงต้องมีอัลกอริทึมหรือวิธีการมาช่วยในการแก้ปัญหาว่าค่าน้ำหนักรวมที่ได้เป็นค่าที่น้อยที่สุดจริงหรือไม่ ในบทนี้จะอธิบายอัลกอริทึมของครูสกาลและอัลกอริทึมของพริม สำหรับ หาทรีแบบทอดข้ามที่เล็กที่สุด ซึ่งจะกล่าวรายละเอียดในหัวข้อถัดไปดังนี้

(1) อัลกอริทึมของครูสกาล

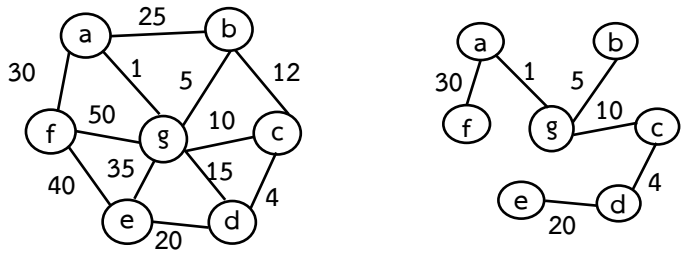
อัลกอริทึมของครูสกาล(Kruskal's Algorithm) เป็นอัลกอริทึมที่นำเส้นเชื่อมที่มีน้ำหนักน้อยที่สุดมาต่อกันทีละเส้นโดยต้องไม่ทำให้เกิดเส้นทางแบบไซเคิล หากทำจนครบทุกจุดยอดในกราฟแล้วจะได้ทรีแบบทอดข้ามที่เล็กที่สุด จะเห็นว่าต้องพิจารณาทีละเส้นเชื่อมที่มีน้ำหนักน้อยที่สุดมาประกอบรวมกันดังนั้นก่อนเริ่มต้นหาทรีแบบทอดข้ามที่เล็กที่สุด จึงต้องนำเส้นเชื่อมมาเรียงลำดับตามค่าน้ำหนักจากน้อยไปมาก โดยใช้โครงสร้างข้อมูลคิวแบบลำดับความสำคัญ (Priority Queue) มาช่วยในการจัดเก็บข้อมูลซึ่งมีตัวอย่างอัลกอริทึมดังนี้

```
KrusalAlgo(G)
  Create an emptyp graph T
  Create a priority queue PQ
  containing all edges
  VS = ∅
  for each vertex v ∈ V do add {v} to VS
  While | VS | > 1
      (v,w) =PQ.dequeue()
      if v and w are in different set w1 and w2 in VS then
        add (v,w) to T
      replace w1 and w2 in VS by w1 ∪ w2
```

สร้างโครงสร้างกราฟ T และสร้างคิวแบบลำดับความสำคัญ (PQ) เก็บข้อมูลเส้นเชื่อมที่เรียงลำดับตามค่าน้ำหนักของเส้นเชื่อมจากน้อยไปมาก

สร้าง VS เก็บชุดเซตของจุดยอดทั้งหมดในกราฟ

- นำแต่ละเส้นเชื่อม (v,w) ออกจาก PQ ทีละเส้น โดยเลือกเส้นเชื่อมที่มีค่าน้ำหนักน้อยที่สุดก่อน
- หากเป็นเส้นเชื่อมที่ไม่ทำให้เกิดไซเคิล (v และ w อยู่ต่างเซตกัน) ให้เพิ่มเส้นเชื่อมนี้ในโครงสร้าง T



(ก) ตัวอย่างกราฟ **(ข) ทรีแบบทอดข้ามที่เล็กที่สุด**

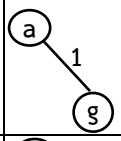
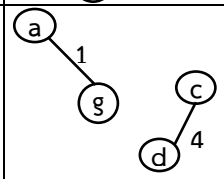
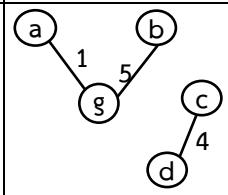
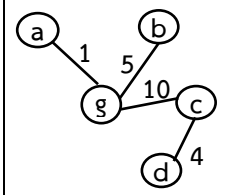
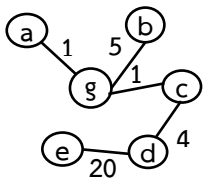
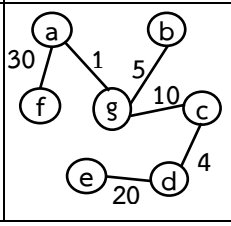
รูปที่ 11.9 ตัวอย่างทรีแบบทอดข้ามที่เล็กที่สุด

จากตัวอย่างกราฟในรูปที่ 11.9 สามารถแสดงลำดับการหาทรีแบบทอดข้ามที่น้อยที่สุดด้วยวิธีครูสกาลได้ดังตารางที่ 11.4 โดยจากหลักการทำงานของอัลกอริทึมของครูสกาลข้างต้น ประกอบด้วยการจัดเก็บข้อมูล ดังนี้

1) โครงสร้างคิวแบบลำดับความสำคัญ PQ ที่เก็บเส้นเชื่อมเรียงลำดับตามลำดับความสำคัญของเส้นเชื่อมโดยเส้นเชื่อมที่มีน้ำหนักน้อยจะมีความสำคัญกว่าเส้นเชื่อมที่มีน้ำหนักมาก ดังนั้นตัวอย่างในแถวที่ 1 ของตารางที่ 11.4 พบว่าเริ่มต้น PQจะเก็บค่าคู่ลำดับของจุดยอดดังนี้

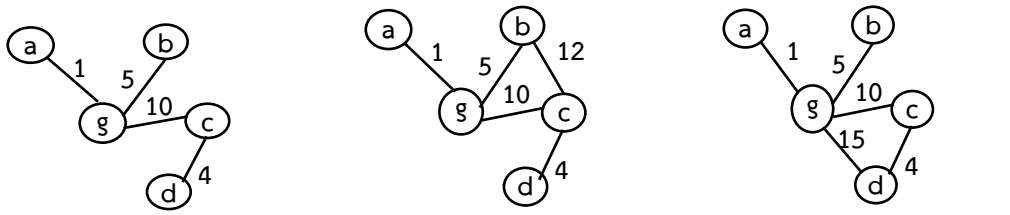
(a,g), (c,d), (b,g), (c,g), (b,c), (d,g), (d,e), (a,b), (e,f), (a,e), (e,g), (e,f)

ตารางที่ 11.4 ลำดับการหาทรีแบบทอดข้ามที่น้อยที่สุดด้วยวิธีครูสกาล

ลำดับ	การเลือกเส้นเชื่อม (v,w) ใน PQ	ค่าใน VS	โครงสร้างกราฟ T	ค่าใน PQ
-		{a}, {b}, {c}, {d}, {e}, {f}, {g}	ว่าง	(a,g), (c,d), (b,g), (c,g), (b,c), (d,g), (d,e), (a,b), (e,f), (a,e), (e,g), (e,f)
1	- นำเส้นเชื่อม (a,g) ออกจากคิว (ค่าน้ำหนักเท่ากับ 1) - เพิ่มในโครงสร้าง T	{a, g}, {b}, {c}, {d}, {e}, {f}		(c,d), (b,g), (c,g), (b,c), (d,g), (d,e), (a,b), (e,f), (a,e), (e,g), (e,f)
2	- นำเส้นเชื่อม (c,d) ออกจากคิว (ค่าน้ำหนักเท่ากับ 4) - เพิ่มในโครงสร้าง T	{a, g}, {b}, {c, d}, {e}, {f}		(b,g), (c,g), (b,c), (d,g), (d,e), (a,b), (e,f), (a,e), (e,g), (e,f)
3	- นำเส้นเชื่อม (b,g) ออกจากคิว (ค่าน้ำหนักเท่ากับ 5) - เพิ่มในโครงสร้าง T	{a, b, g}, {c, d}, {e}, {f}		(c,g), (b,c), (d,g), (d,e), (a,b), (e,f), (a,e), (e,g), (e,f)
4	- นำเส้นเชื่อม (c,g) ออกจากคิว (ค่าน้ำหนักเท่ากับ 10) - เพิ่มในโครงสร้าง T	{a, b, c, d, g}, {e}, {f}		(b,c), (d,g), (d,e), (a,b), (e,f), (a,e), (e,g), (e,f)
5	- นำเส้นเชื่อม (b,c) ออกจากคิว (ค่าน้ำหนักเท่ากับ 12) - เป็นเส้นเชื่อมที่ทำให้เกิดไซเคิล	ไม่เปลี่ยนแปลง	ไม่เปลี่ยนแปลง	(d,g), (d,e), (a,b), (e,f), (a,e), (e,g), (e,f)
6	- นำเส้นเชื่อม (d,g) ออกจากคิว (ค่าน้ำหนักเท่ากับ 15) - เป็นเส้นเชื่อมที่ทำให้เกิดไซเคิล	ไม่เปลี่ยนแปลง	ไม่เปลี่ยนแปลง	(d,e), (a,b), (e,f), (a,e), (e,g), (e,f)
7	- นำเส้นเชื่อม (d,e) ออกจากคิว (ค่าน้ำหนักเท่ากับ 20) - เพิ่มในโครงสร้าง T	{a, b, c, d, e, g}, {f}		(a,b), (e,f), (a,e), (e,g), (e,f)
8	- นำเส้นเชื่อม (a,b) ออกจากคิว (ค่าน้ำหนักเท่ากับ 25) - เป็นเส้นเชื่อมที่ทำให้เกิดไซเคิล	ไม่เปลี่ยนแปลง	ไม่เปลี่ยนแปลง	(e,f), (a,e), (e,g), (e,f)
9	- นำเส้นเชื่อม (a,f) ออกจากคิว (ค่าน้ำหนักเท่ากับ 30) - เพิ่มในโครงสร้าง T	{a, b, c, d, e, f, g}		(a,e), (e,g), (e,f)

2) โครงสร้างกราฟ T สำหรับจัดเก็บทรีแบบทอดข้ามที่เล็กที่สุด โดยเริ่มต้นจะเป็นกราฟว่าง เมื่อนำเส้นเชื่อมที่มีน้ำหนักน้อยที่สุดจาก PQ มาเพิ่มในโครงสร้าง Tแบบไม่ให้เกิดไซเคิล ดังตัวอย่างในตารางที่ 11.4 พบว่าแถวที่ 1-4,7,9 เป็นตัวอย่างการเพิ่มเส้นเชื่อมในโครงสร้าง T แบบไม่เกิดไซเคิล ส่วนกรณีแถวที่ 5-6 และ 8 เป็นตัวอย่างเส้นเชื่อมที่หากเพิ่มในโครงสร้าง T จะทำให้เกิดไซเคิล จึงไม่เพิ่มในโครงสร้าง T เมื่อเพิ่มเส้นเชื่อมในโครงสร้าง T จนครบทุกจุดยอดในกราฟ จะได้โครงสร้างกราฟที่มีลักษณะเฉพาะคือเป็นทรีแบบทอดข้ามที่น้อยที่สุด ดังรูปที่ 11.9 (ข) ซึ่งเป็นทรีแบบทอดข้ามที่น้อยที่สุดด้วยค่าน้ำหนักรวมเท่ากับ 70

1) โครงสร้าง VS สำหรับเก็บเซตของจุดยอดทั้งหมดในกราฟ โดยเริ่มต้นขนาดของ VS จะเท่ากับจำนวนจุดยอดในกราฟเมื่อสิ้นสุดการทำงานของอัลกอริทึมในการหา ทรีแบบทอดข้ามที่เล็กที่สุด ขนาด VS จะมีค่าเป็น 1 โดยข้อมูลที่จัดเก็บใน VS จะถูกใช้เพื่อตรวจสอบว่าเส้นเชื่อม (v,w) ที่จะเพิ่มในโครงสร้าง T ว่าจะทำให้เกิดไซเคิลหรือไม่ โดยหากจุดยอด v และ w อยู่ในเซตเดียวกันในโครงสร้าง VS อยู่แล้วแสดงว่าหากเพิ่มเส้นเชื่อม (v,w) จะทำให้เกิดไซเคิล ดังตัวอย่างในรูปที่ 11.10 (ก) ค่า VS = {a, b, c, d, g}, {e}, {f} หากเพิ่มเส้นเชื่อม (b,c) ใน T จะเกิดไซเคิลดังรูปที่ 11.10 (ข) และหากเพิ่มเส้นเชื่อม (d,g) ใน T จะเกิดไซเคิลดังรูปที่ 11.10 (ค) ซึ่งพบว่าจุดยอด b และ c และจุดยอด d และ g ต่างเป็นคู่จุดยอดที่อยู่ในเซตเดียวกันใน VS อยู่แล้วทั้งสองกรณี



(ก) ตัวอย่างโครงสร้างกราฟ T **(ข) เมื่อเพิ่มเส้นเชื่อม (b,c)** **(ค) เมื่อเพิ่มเส้นเชื่อม (d,g)**

รูปที่ 11.10 ตัวอย่างทรีแบบทอดข้าม

(2) อัลกอริทึมของพริม

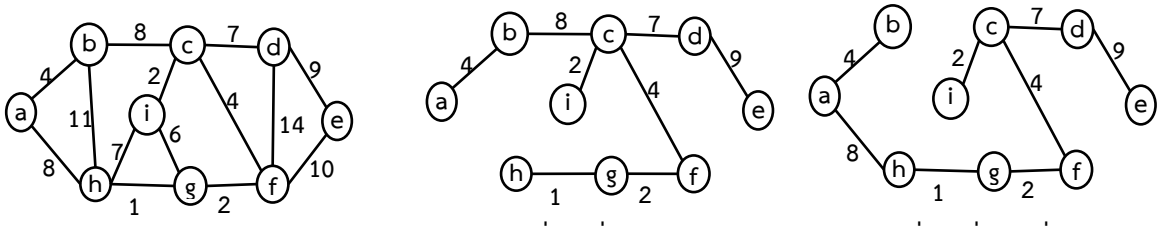
อัลกอริทึมของพริม (Prim's Algorithm) เป็นอัลกอริทึมที่นำเส้นเชื่อมที่มีน้ำหนักน้อยที่สุดอีกวิธีหนึ่ง โดยมีการประยุกต์ใช้โครงสร้างข้อมูลคิวแบบลำดับความสำคัญ (Priority Queue) มาช่วยในการจัดเก็บข้อมูล เช่นเดียวกับอัลกอริทึมของครูสกาล แต่ต่างกันที่จะเริ่มต้นจากโหนดใดโหนดหนึ่งก่อน แล้วนำเส้นเชื่อมเข้าจัดเก็บในคิว โดยต้องเป็นเส้นเชื่อมที่ไม่ทำให้เกิดจุดยอดซ้ำกันในคิว ตัวอย่างอัลกอริทึมของพริมมีดังนี้

```
T = a spanning tree containing a single node s;
PQ = set of edges adjacent to s;
while T does not contain all the nodes {
  remove an edge (v, w) of lowest cost from E
  if w is already in T  then discard edge (v, w)
  else
        add edge (v, w) and node w to T
        add to E the edges adjacent to w
  }
}
```

จากอัลกอริทึมข้างต้น สามารถแสดงตัวอย่างการหาทรีแบบทอดข้ามที่น้อยที่สุดด้วยวิธีพริมจากโครงสร้างกราฟในรูปที่ 11.9 ได้ดังตารางที่ 11.5 โดยจากหลักการทำงานของอัลกอริทึมของพริมข้างต้น ประกอบด้วยการจัดเก็บข้อมูล ดังนี้

1) โครงสร้างคิวแบบลำดับความสำคัญ PQ จะเก็บค่าเส้นเชื่อมตามลำดับความสำคัญของน้ำหนักเส้นเชื่อม โดยเส้นเชื่อมที่มีน้ำหนักน้อยจะมีความสำคัญกว่าเส้นเชื่อมที่มีน้ำหนักมาก ดังนั้นตัวอย่างในแถวที่ 1 ของตารางที่ 11.5 พบว่าเริ่มต้น PQ จะเก็บค่าคู่ลำดับของจุดยอด a (กำหนดเป็นจุดยอดเริ่มต้น) คือ (a,g), (a,b), (a,f)
2) โครงสร้าง T ใช้จัดเก็บทรีแบบทอดข้ามที่เล็กที่สุด โดยเริ่มต้นด้วยจุดยอดที่กำหนดเป็นจุดยอดเริ่มต้น ดังตัวอย่างในแถวที่ 1 ของตารางที่ 11.5 กำหนด a เป็นจุดยอดเริ่มต้น จากนั้น
   2.1)เพิ่มจุดยอดถัดไป (w) ในโครงสร้าง T ด้วยเส้นเชื่อม (v,w) ที่มีค่าน้ำหนักน้อยที่สุดใน PQ ดังตัวอย่างในแถวที่ 2 ของตารางที่ 11.5 จะเพิ่มจุดยอด g ใน T เนื่องจากมีค่าน้ำหนักเส้นเชื่อม (a,g) น้อยสุด คือ 1
   2.2)นำทุกเส้นเชื่อม (w,x) ระหว่างจุดยอด w กับจุดยอดใกล้เคียงของ w เข้า PQ โดยจะต้องเป็นจุดยอด xที่ยังไม่ปรากฏในกราฟ T และไม่ใช่จุดยอดซ้ำใน PQ ดังตัวอย่างในแถวที่ 2 ของตารางที่ 11.5 พบว่าจุดยอดใกล้เคียงทั้งหมดของ g คือ a b c d e และ f ซึ่งมีการพิจารณานำเส้นเชื่อมเข้าคิวดังนี้
      - a เป็นจุดยอดใน T ดังนั้นเส้นเชื่อม (a,g) จึงไม่ถูกนำเข้าคิว
      - b เป็นจุดยอดที่อยู่ใน PQ แล้วแต่เนื่องจากน้ำหนักเส้นเชื่อม (b,g) มีค่าน้อยกว่าเส้นเชื่อม (a,b) ดังนั้นจึงนำ (a,b) ออกจากคิว และนำ (b,g) เข้าคิวแทน
      - c, d และ e เป็นจุดยอดที่ไม่ได้อยู่ใน T และไม่อยู่ใน PQ จึงนำเส้นเชื่อม (c,g), (d,g) และ (e,g) เข้าคิว
      - f เป็นจุดยอดที่อยู่ใน PQ แล้ว และน้ำหนักเส้นเชื่อม (a,f) มีค่าน้อยกว่าเส้นเชื่อม (f,g) ดังนั้นจึงไม่นำ (f,g) เข้าคิว
3) ให้วนทำซ้ำข้อ 2) เพื่อนำจุดยอดต่าง ๆ ที่มีค่าน้ำหนักน้อยที่สุดจาก PQ ใส่เพิ่มใน T ไปเรื่อยๆ ดังลำดับในแถวที่ 3-7 ในตารางที่ 11.5 จนกระทั่งครบทุกจุดยอดในกราฟ จะได้ทรีแบบทอดข้ามที่น้อยที่สุดด้วยค่าน้ำหนักรวมเท่ากับ 70 ซึ่งพบว่าจะได้โครงสร้างทรีแบบทอดข้ามที่น้อยที่สุดเหมือนกับอัลกอริทึมของครูสกาลในตารางที่ 11.4

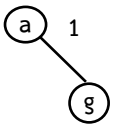
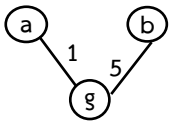
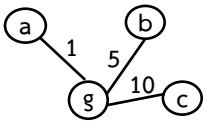
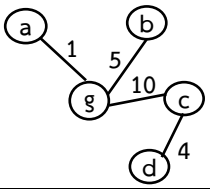
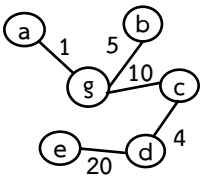
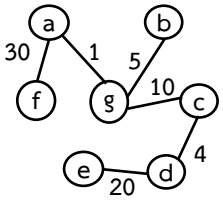
อย่างไรก็ตามโครงสร้างกราฟที่มีน้ำหนักของเส้นเชื่อมซ้ำกัน อาจหาโครงสร้างทรีแบบทอดข้ามที่น้อยที่สุดได้หลายรูปแบบ ดังตัวอย่างรูปที่ 11.11 พบว่าโครงสร้างกราฟในรูปที่ 11.11 (ก) เป็นกราฟแบบมีเส้นเชื่อมที่มีค่าน้ำหนักซ้ำกัน ทำให้สามารถหา โครงสร้างทรีแบบทอดข้ามที่น้อยที่สุดได้มากกว่าหนึ่งรูปแบบดังรูปที่ 11.11 (ข) และ (ค) ซึ่งมีค่าน้ำหนักรวมเท่ากัน คือ 37



(ก) ตัวอย่างโครงสร้างกราฟ (ข) ทรีแบบทอดข้ามที่น้อยที่สุดรูปแรก (ค) ทรีแบบทอดข้ามที่น้อยที่สุดรูปที่สอง

รูปที่ 11.11 ตัวอย่างโครงสร้างกราฟที่หาทรีแบบทอดข้ามที่น้อยที่สุดได้มากกว่าหนึ่งรูปแบบ

ตารางที่ 11.5 ลำดับการหาทรีแบบทอดข้ามที่น้อยที่สุดด้วยวิธีพริม

ลำดับ	การเลือกเส้นเชื่อม (v,w) ใน PQ	โครงสร้างกราฟ T	ค่าใน PQ
1	- เริ่มต้นด้วยจุดยอด a - นำเส้นเชื่อมระหว่าง a กับ adjacent ของ a เข้าคิว		(a,g), (a,b), (a,f)
2	- นำจุดยอด g ที่มีค่าน้ำหนักเส้นเชื่อม (a,g) น้อยที่สุด ออกจากคิว (ค่าน้ำหนักเท่ากับ 1) เพิ่มในโครงสร้าง T - พิจารณาเส้นเชื่อมระหว่าง g กับ adjacent ของ g เข้าคิว (มีการนำ (a,b) ออกจากคิว และนำ (b,g) เข้าคิวแทน)		(b,g), (c,g), (d,g), (e,g), (a,f)
3	- นำจุดยอด b ที่มีค่าน้ำหนักเส้นเชื่อม (b,g) น้อยที่สุด ออกจากคิว (ค่าน้ำหนักเท่ากับ 5) เพิ่มในโครงสร้าง T - พิจารณาเส้นเชื่อมระหว่าง b กับ adjacent ของ b เข้าคิว (เป็นกรณีที่ไม่มีการเปลี่ยนแปลงค่าในคิว)		(c,g), (d,g), (e,g), (a,f)
4	- นำจุดยอด c ที่มีค่าน้ำหนักเส้นเชื่อม (c,g) น้อยที่สุดออกจากคิว (ค่าน้ำหนักเท่ากับ 10) เพิ่มในโครงสร้าง T - พิจารณาเส้นเชื่อมระหว่าง c กับ adjacent ของ c เข้าคิว (มีการนำ (d,g) ออกจากคิว และนำ (c,d) เข้าคิวแทน)		(c,d), (e,g), (a,f)
5	- นำจุดยอด d ที่มีค่าน้ำหนักเส้นเชื่อม (c,d) น้อยที่สุดออกจากคิว (ค่าน้ำหนักเท่ากับ 4) เพิ่มในโครงสร้าง T - พิจารณาเส้นเชื่อมระหว่าง d กับ adjacent ของ d เข้าคิว (เป็นกรณีที่ไม่มีการเปลี่ยนแปลงค่าในคิว)		(e,g), (a,f)
6	- นำจุดยอด e ที่มีค่าน้ำหนักเส้นเชื่อม (d,e) น้อยที่สุดออกจากคิว (ค่าน้ำหนักเท่ากับ 20)) เพิ่มในโครงสร้าง T - พิจารณาเส้นเชื่อมระหว่าง e กับ adjacent ของ e เข้าคิว (เป็นกรณีที่ไม่มีการเปลี่ยนแปลงค่าในคิว)		(a,f)
7	- นำจุดยอด f ที่มีค่าน้ำหนักเส้นเชื่อม (a,f)น้อยที่สุดออกจากคิว(ค่าน้ำหนักเท่ากับ 30)เพิ่มในโครงสร้าง T - พิจารณาเส้นเชื่อมระหว่าง d กับ adjacent ของ d เข้าคิว		ว่าง

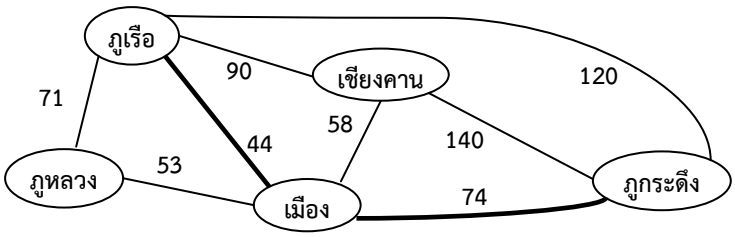
ปัญหาการหาทรีแบบทอดข้ามที่เล็กที่สุดสามารถนำไปประยุกต์ใช้เพื่อแก้ปัญหาในชีวิตประจำวันต่าง ๆ มากมาย เช่น

1) การเดินสายไฟภายในบ้านแบบใดที่สามารถเชื่อมต่อทุกจุดภายในบ้านแล้วใช้สายไฟน้อยที่สุด
2) การใช้สายเคเบิลน้อยที่สุดในการเชื่อมต่อคอมพิวเตอร์เป็นระบบเครือข่าย
3) การสร้างสวนสนุก โดยให้เสียค่าใช้จ่ายน้อยที่สุดในการกำหนดเส้นทางในสวนสนุกให้สามารถเดินติดต่อถึงกันถึงทุกจุดของเครื่องเล่นในสวนสนุก

11.4.5 เส้นทางที่สั้นที่สุด

ปัญหาพื้นฐานสำคัญเรื่องหนึ่งในโครงสร้างข้อมูลกราฟ คือ การหาเส้นทางที่สั้นที่สุดในกราฟที่มีน้ำหนักซึ่งเป็นปัญหาที่พบทั่วไปในชีวิตประจำวัน เช่น ปัญหาการหาระยะทางหรือเวลาที่น้อยที่สุดในการเดินทางระหว่างเมืองหรือจังหวัดต่างๆ ปัญหาการส่งของให้เสียค่าใช้จ่ายน้อยที่สุดในระบบขนส่งสินค้าของบริษัท ปัญหาการส่งข้อมูลไปตามสายใยแก้วนำแสงให้เร็วและเสียค่าใช้จ่ายน้อยที่สุดในระบบเครือข่ายคอมพิวเตอร์เป็นต้น สำหรับหัวข้อนี้จะสนใจเฉพาะการแก้ปัญหาการหาเส้นทางที่สั้นที่สุดจากจุดเดียว (SingleSource Shortest Path)ด้วยอัลกอริทึมของไดก์สตรา (Dijkstra's Algorithm) ด้วยข้อจำกัดที่ว่าค่าน้ำหนักของเส้นเชื่อมต้องมีค่าที่ไม่ติดลบ (Non Negative) โดยจะเป็นกราฟแบบมีทิศทางหรือไม่ก็ได้ แต่จะต้องเป็นกราฟแบบต่อเนื่อง (Connected Graph) คือ มีเส้นทางระหว่างทุกๆจุดยอดในกราฟ

อัลกอริทึมของไดก์สตราเป็นวิธีการในการหาค่าระยะทาง (Distance) ที่เป็นเส้นทางที่สั้นที่สุด (Shortest Path) จากจุดยอดต้นทาง (Source) (เรียกจุดยอด s)ไปยังจุดยอดใดๆ (เรียกจุดยอด v)ในกราฟ ดังตัวอย่างกราฟแสดงระยะทางระหว่างเมืองต่าง ๆ ในรูปที่ 11.12 สามารถหาเส้นทางระหว่างอำเภอภูเรือไปอำเภอภูกระดึงได้หลายเส้นทางด้วยกัน เช่น ภูเรือ-ภูหลวง-เมือง-ภูกระดึง ภูเรือ-ภูหลวง-เชียงคาน-ภูกระดึง ภูเรือ-เมือง-ภูกระดึง ภูเรือ-เชียงคาน-ภูกระดึง และภูเรือ-ภูกระดึง เป็นต้น แต่จะพบว่าเส้นทางที่สั้นที่สุด จะหมายถึงเส้นทาง ภูเรือ-เมือง-ภูกระดึง ด้วยผลรวมระยะทางสั้นที่สุด คือ 118 เป็นต้น



รูปที่ 11.12 ตัวอย่างเส้นทางที่สั้นที่สุด

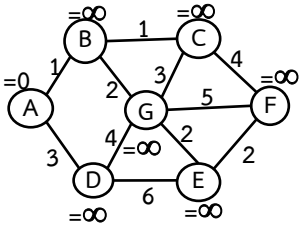
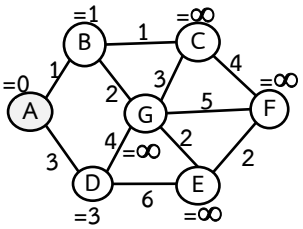
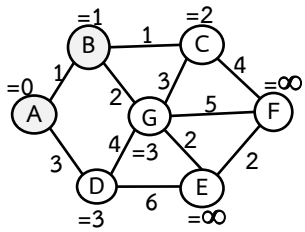
การหาเส้นทางที่สั้นที่สุดจากจุดยอดต้นทาง s ไปยังจุดยอดปลายทาง t ใดจะให้ค่าน้ำหนักรวมของแต่ละเส้นเชื่อมบนเส้นทางมีค่าน้อยที่สุดดังตัวอย่างลำดับคำสั่งต่อไปนี้

```
For all v∈V do
d[v] <- ∞
prev[v] = null
PQ <- V[G]
while Q ≠∅ do {
        u <- Extract-Min(PQ)
        for each v ∈ Adj[u] do {
        if d[v] > d[u]+w(u, v) {
                d[v] <- d[u]+ w(u, v)
                pred[v] <- u
          }
        }
}
```

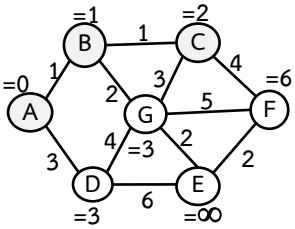
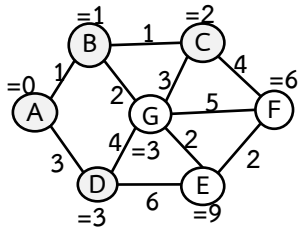
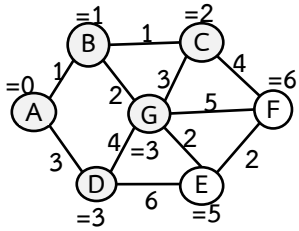
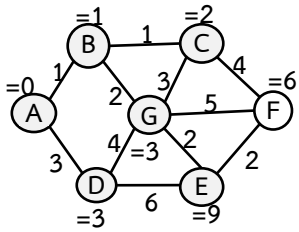
จากอัลกอริทึมไดก์สตราสามารถแสดงตัวอย่างขั้นตอนการหาเส้นทางสั้นที่สุดได้ดังตารางที่ 11.6 ด้วยแนวความคิดที่สำคัญของอัลกอริทึมไดก์สตรา คือยิ่งเดินทางผ่านจุดยอดจำนวนมากเท่าไรก็ยิ่งได้ค่าน้ำหนักรวมมากขึ้นเช่นกัน เพราะค่าน้ำหนักในเส้นเชื่อมไม่มีค่าติดลบ ดังนั้น

1) กำหนดระยะทาง d[v] เพื่อเก็บค่าระยะทางที่สั้นที่สุดจากจุดยอดเริ่มต้น s ไปยังจุดยอด v โดยเริ่มต้นให้กับแต่ละจุดยอด v มี d[v] เท่ากับ ∞ ยกเว้น d[s] จะเท่ากับ 0 และกำหนด pred[v] = null สำหรับเก็บข้อมูลว่าเส้นเทางจากจุดยอดเริ่มต้นมายังจุดยอด v ต้องผ่านจุดยอดใดมาก่อนก่อนหน้าจุดยอด v
2) ให้พิจารณาจุดยอด u ที่มีค่า d[u] น้อยที่สุดจากนั้นหาจุดยอดใกล้เคียง (เรียก v) ของจุดยอด u โดยแต่ละจุดยอด v ให้พิจารณาค่าระยะทางของเส้นทางเดิม d[v] กับค่าระยะทางของเส้นทางใหม่ คือ d[v] = d[u]+w(u,v) ซึ่งเป็นระยะทาง d[u] คือ ระยะทางจากจุดยอดเริ่มต้นไปยังจุดยอด u บวกกับระยะทางจากจุดยอด u มายังจุดยอด v รวมเป็นระยะทางใหม่ d[v] ของจุดยอด v โดยให้เลือกเส้นทางที่ทำให้ได้ระยะทาง d[v] ที่น้อยกว่าและถือว่าเป็นเส้นทางที่สั้นที่สุดจากจุดยอดเริ่มต้น s ไปยังจุดยอด v
3) วนทำซ้ำข้อ 2 จนกระทั่งได้เส้นทางที่สั้นที่สุดจากจุดยอดต้นทาง s ไปยังทุกจุดยอดปลายทาง t ในกราฟ

ตารางที่ 11.6 ลำดับการหาเส้นทางที่สั้นที่สุดด้วยอัลกอริทึมไดก์สตรา

ลำดับ	u	v	คำอธิบายการเลือกเส้นทางสั้นที่สุด	การเปลี่ยนแปลงค่า d[v]	ค่า pred	PQ
1	-	-	-กำหนดระยะทางเริ่มต้นให้กับแต่ละจุดยอด = ∞ -กำหนดระยะทางจุดยอดเริ่มต้น s เป็น 0 ในกรณีนี้คือ จุดยอด A ดังนั้น d[A] = 0		pred[A]=null pred[B]=null pred[C]=null pred[D]=null pred[E]=null pred[F]=null pred[G]=null	{A B C D E F G}
2	A	B D	- จุดยอดใกล้เคียงของ A คือ B และ D - คำนวณระยะทางใหม่ของ B และ D - d[A] + w(A,B) =0+1=1 - d[A] + w(A,D) =0+3=3 - เลือกเส้นทางที่ระยะทางสั้นสุด - d[B]=min(∞,0+1)=1 - d[D]=min(∞,0+3)=3		pred[A]=null pred[B]=A pred[C]=null pred[D]=A pred[E]=null pred[F]=null pred[G]=null	{B C D E F G}
3	B	C G	- จุดยอดใกล้เคียงของ Bคือ C และ G - คำนวณระยะทางใหม่ของ C และ G - d[B] + w(B,C) =1+1=2 - d[B] + w(B,G) =1+2=3 - เลือกเส้นทางที่ระยะทางสั้นสุด - d[C]=min(∞,1+1)=2 - d[G]=min(∞,1+2)=3		pred[A]=null pred[B]=A pred[C]=B pred[D]=A pred[E]=null pred[F]=null pred[G]=B	{C D G E F}

ตารางที่ 11.6 ลำดับการหาเส้นทางที่สั้นที่สุดด้วยอัลกอริทึมไดก์สตรา (ต่อ)

ลำดับ	u	v	คำอธิบายการเลือกเส้นทางสั้นที่สุด	การเปลี่ยนแปลงค่า d[v]	ค่า pred	PQ
4	C	F G	- จุดยอดใกล้เคียงของ Cคือ F และ G - คำนวณระยะทางใหม่ของ F และ G - d[C] + w(C,F) =2+4=6 - d[C] + w(C,G) =2+3=5 - เลือกเส้นทางที่ระยะทางสั้นสุด - d[F]=min(∞,2+4)=6 - d[G]=min(3,2+3)=3		pred[A]=null pred[B]=A pred[C]=B pred[D]=A pred[E]=null pred[F]=C pred[G]=B	{D G F E}
5	D	E G	- จุดยอดใกล้เคียงของ Dคือ E และ G - คำนวณระยะทางใหม่ของ E และ G - d[D] + w(D,E) =3+6=9 - d[D] + w(D,G) =3+4=7 - เลือกเส้นทางที่ระยะทางสั้นสุด - d[E]=min(∞,3+6)=9 - d[G]=min(3,3+4)=3		pred[A]=null pred[B]=A pred[C]=B pred[D]=A pred[E]=D pred[F]=C pred[G]=B	{G F E}
6	G	E F	- จุดยอดใกล้เคียงของ Gคือ E และ F - คำนวณระยะทางใหม่ของ E และ F - d[G] + w(G,E) =3+2=5 - d[G] + w(G,F) =3+5=8 - เลือกเส้นทางที่ระยะทางสั้นสุด - d[E]=min(9,3+2)=5 - d[F]=min(6,3+5)=6		pred[A]=null pred[B]=A pred[C]=B pred[D]=A pred[E]=G pred[F]=C pred[G]=B	{E F}
7	E	F	- จุดยอดใกล้เคียงของ Eคือ F - คำนวณระยะทางใหม่ของ F - d[E] + w(E,F) = 5+2=7 - เลือกเส้นทางที่ระยะทางสั้นสุด - d[F]=min(6,5+2)=6		pred[A]=null pred[B]=A pred[C]=B pred[D]=A pred[E]=G pred[F]=C pred[G]=B	{F}

จากตารางที่ 11.6 สามารถอธิบายตัวอย่างการเลือกเส้นทางที่สั้นที่สุด ได้ดังนี้

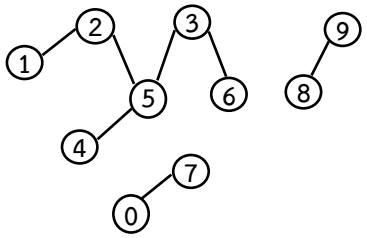
- จากแถวที่ 2 ในตารางที่ 11.6 พบว่าการหาระยะทางสั้นที่สุดจากจุดยอด u ไปยังจุดยอดใกล้เคียง v ของ u (ในที่นี้ u คือ จุดยอด A และ v คือ จุดยอด B และ D) ซึ่งพบว่า ค่าระยะ d[B] เดิม คือ ∞ ส่วนระยะทางของเส้นทางใหม่ คือ d[B] = d[A]+w(A,B) = 0 + 1 = 1 และค่าระยะ d[D] เดิม คือ ∞ ส่วนระยะทางของเส้นทางใหม่ คือ d[D] = d[A]+w(A,D) = 0 + 3 = 3 เช่นกัน นั่นคือจะเลือกเส้นทางใหม่ทั้งคู่ และกำหนดค่า pred[B] = Aและค่า pred[D] = A ซึ่งถือเป็นเส้นทางที่สั้นที่สุดจากจุดยอดต้นทาง s (จุดยอด A) ไปยังจุดยอด v (จุดยอด B,D) ใดๆ
- จากแถวที่ 4 ในตารางที่ 11.6 พบว่าการหาระยะทางสั้นที่สุดจากจุดยอด u ไปยังจุดยอดใกล้เคียง v ของ u (ในที่นี้ u คือ จุดยอด C และ v คือ จุดยอด F และ G) ซึ่งพบว่า ค่าระยะ d[F] เดิม คือ ∞ ส่วนระยะทางของเส้นทางใหม่ คือ d[F] = d[C]+w(C,F) = 2 + 4 = 6

ดังนั้นจะเลือกเส้นทางใหม่ และกำหนดค่า pred[F] = C และค่าระยะ d[G] เดิม คือ 3 แต่เส้นทางใหม่ คือ d[G] = d[C]+w(C,G) = 2 + 3 = 5 ดังนั้นจึงเลือกเส้นทางเดิม

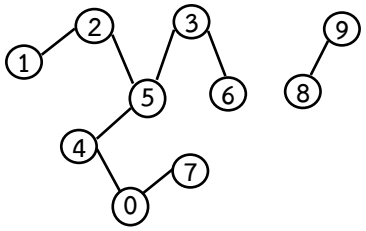
- จากแถวที่ 6 ในตารางที่ 11.6 พบว่าการหาระยะทางสั้นที่สุดจากจุดยอด u ไปยังจุดยอดใกล้เคียง v ของ u (ในที่นี้ u คือ จุดยอด G และ v คือ จุดยอด Eและ F) พบว่าค่าระยะ d[E] เดิม คือ 9 ส่วนเส้นทางใหม่ คือ d[E] = d[G]+w(G,F) = 2 + 4 = 6 ดังนั้นจะเลือกเส้นทางใหม่ และกำหนดค่า pred[E] = G นั่นคือเส้นทางที่สั้นที่สุดจากจุดยอด A มาจุดยอด E จะต้องเลือกผ่านจุดยอด G แทนจุดยอด D (ค่า pred[E] เดิม คือ D) และค่าระยะ d[F] เดิม คือ 6 แต่เส้นทางใหม่ คือ d[F] = d[G]+w(G,F) = 2 + 5 = 7 ดังนั้นจึงเลือกเส้นทางเดิม

11.4.6 การประยุกต์ใช้งานโครงสร้างเซตไม่มีส่วนร่วมในโครงสร้างกราฟ

โครงสร้างข้อมูลเซตไม่มีส่วนร่วมเป็นโครงสร้างข้อมูลที่ใช้เก็บข้อมูลที่ถูกแบ่งเป็นหลายๆ เซต โดยที่แต่ละเซตไม่มีส่วนร่วมกันเลย ซึ่งสามารถนำมาประยุกต์ใช้งานร่วมกับโครงสร้างข้อมูลกราฟได้ ในหัวข้อนี้จะอธิบายถึงตัวอย่างการใช้โครงสร้างเซตไม่มีส่วนร่วม ในการเก็บข้อมูลของโครงสร้างกราฟแบบไม่ต่อเนื่อง ดังรูปที่ 11.14 เป็นโครงสร้างกราฟที่แยกเป็น 3 เซตแบบไม่มีส่วนร่วม ดังรูปที่ 11.14 (ก) ซึ่งเซตที่ 1 ประกอบด้วยจุดยอด 1 2 3 4 5 และ 6 เซตที่สองประกอบด้วยจุดยอด 0 และ 7 ส่วนเซตที่ 3 ประกอบด้วยจุดยอด 8 และ 9 โดยหากต้องการรวมเซตที่ 1 และเซตที่ 2 เข้าด้วยกันด้วยคำสั่งยูเนียน จะได้กราฟที่ประกอบด้วย 2 เซต ดังรูปที่ 11.14 (ข)



(ก) โครงสร้างกราฟที่ประกอบด้วย 3 เซต



(ก) โครงสร้างกราฟที่ประกอบด้วย 3 เซต

รูปที่ 11.14 กราฟแบบไม่ต่อเนื่องที่มีลักษณะเป็นโครงสร้างเซตไม่มีส่วนร่วม

จากตัวอย่างการจัดเก็บกราฟแบบไม่ต่อเนื่องด้วยโครงสร้างเซตไม่มีส่วนร่วมในรูปที่ 11.14 จะเห็นตัวอย่างการประยุกต์การใช้งานได้จากการหาทรีแบบทอดข้ามที่น้อยที่สุดด้วยวิธีครูสกาลในหัวข้อก่อนหน้านี้ซึ่งจะพบว่าการเพิ่มเส้นเชื่อม (u,v) ในโครงสร้างทรีแบบทอดข้ามได้ก็สามารถทำได้ด้วยการเรียกใช้การดำเนินการค้นหาในการตรวจสอบว่าจุดยอด u และ v ไม่ได้อยู่ในเซตเดียวกัน ส่วนการเพิ่มเส้นเชื่อม (u,v) ในโครงสร้างทรีได้ด้วยการเรียกใช้การดำเนินการยูเนียนนั่นเอง

สำหรับวิธีการในการค้นหาและยูเนียน จะขึ้นอยู่กับลักษณะการจัดเก็บข้อมูล ในหัวข้อนี้จะอธิบายถึงอัลกอริทึมสำหรับการค้นหาและยูเนียน 3 วิธีการ คือ อัลกอริทึมค้นหาแบบเร็วอัลกอริทึมยูเนียนแบบเร็ว และอัลกอริทึมค้นหาแบบเร็ว โดยมีรายละเอียดดังนี้

(1) อัลกอริทึมค้นหาแบบเร็ว

อัลกอริทึมค้นหาแบบเร็ว (Quick-find Algorithm) เป็นวิธีการหนึ่งในการจัดการข้อมูลในโครงสร้างเซตไม่มีส่วนร่วม โดยแต่ละสมาชิกในโครงสร้างเซตจะถูกจัดเก็บในอาร์เรย์ 1 มิติขนาด N ดังรูปที่ 11.15 แสดงตัวอย่างการจัดเก็บสมาชิกของเซตไม่มีส่วนร่วมจำนวน 10 ค่าในอาร์เรย์ id โดย id[i] เก็บ ค่าตัวแทนของเซตของสมาชิกตัวที่ i เมื่อ i = 0 .. N-1, และ d[i] = d[j] เมื่อสมาชิกที่ i และ

สมาชิกที่ j อยู่ในเซตเดียวกัน จากรูปที่ 11.15 จะเห็นว่าสมาชิก 5 และ 6 ถือว่าเป็นสมาชิกที่อยู่ในเซตเดียวกันด้วยตัวแทนของเซต คือ d[i]=6 ส่วนกลุ่มสมาชิก 2 3 4 และ 9 ถือว่าเป็นสมาชิกที่อยู่ในเซตเดียวกันด้วยตัวแทนของเซต คือ d[i]= 9

i	0	1	2	3	4	5	6	7	8	9
id[i]	0	1	9	9	9	6	6	7	8	9

รูปที่ 11.15 โครงสร้างเซตไม่มีส่วนร่วมที่จัดเก็บด้วยอาร์เรย์

การดำเนินการค้นหาสำหรับสมาชิก p และ q ถ้า id[p] และ id[q] เป็นค่าเดียวกันแสดงว่าสมาชิก p และ q อยู่เซตเดียวกันจะคืนค่าจริง ส่วนการดำเนินการยูเนียนจะทำการกำหนดค่า id[x] = id[q] ให้กับทุกสมาชิก xที่มีค่า id[x] เท่ากับ id[p] ดังตัวอย่างคำสั่งภาษาซีต่อไปนี้

```
int id[N];
void QuickFind(){
    for(int i=0; i<id.lenght(); i++)
        id[i] = i;
}
int find(int p, int q){
    return id[p]==id[q];
}
void union(int p, int q){
    int temp = id[p];
    for(int i=0; i<id.lenght(); i++){
        if(id[i]==temp)
                id[i] = id[q];
    }
}
```

- เมื่อเริ่มต้นสร้างโครงสร้างข้อมูลเซตไม่มีส่วนร่วม จะเรียกใช้คำสั่ง QuickFind()สมาชิกแต่ละตัวจะอยู่คนละเซตกัน ดังนั้นจะกำหนดค่า d[i] = i
- เมื่อต้องการให้สมาชิก 5 และ 6 อยู่ในเซตเดียวกันจะเรียกใช้การดำเนินการ union(5,6) จะพบว่า id[5]= id[6] และเมื่อต้องการให้สมาชิก 2 3 4 และ 9 อยู่ในเซตเดียวกัน จะมีการเรียกใช้การดำเนินการ union(2,3) union(3,4) และ union(4,9) ตามลำดับจะได้โครงสร้างข้อมูลเซตไม่มีส่วนร่วมดังรูปที่ 8.15

จากรูปที่ 11.15 เมื่อเรียกใช้การดำเนินการ union(3,6) ซึ่งพบว่า id[3]=9 และ id[6]=6 ดังนั้นจะเปลี่ยนค่าตัวแทนเซตของทุกสมาชิกในเซตเดียวกัน (มีค่า id เท่ากับ 9) ให้เป็นเซตเดียวกับกลุ่มสมาชิกที่มี id เท่ากับ 6 จะได้โครงสร้างข้อมูลเซตไม่มีส่วนร่วมหลังเรียกใช้ union(3,6) ดังรูปที่ 11.16

i	0	1	2	3	4	5	6	7	8	9
id[i]	0	1	6	6	6	6	6	7	8	6

รูปที่ 11.16 โครงสร้างเซตไม่มีส่วนร่วมหลังเรียกใช้ union(3,6) ด้วยอัลกอริทึมค้นหาแบบเร็ว

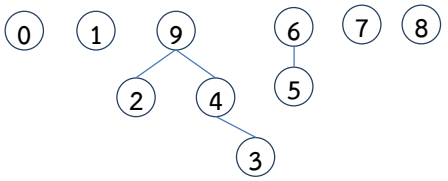
จากรูปที่ 11.15 หากเรียกใช้การดำเนินการ union(6,3) ซึ่งพบว่า id[6]=6 และ id[3]=9 ดังนั้นจะเปลี่ยนค่าตัวแทนเซตของทุกสมาชิกในเซตเดียวกัน (มีค่า id เท่ากับ 6) ให้เป็นเซตเดียวกับกลุ่มสมาชิกที่มี id เท่ากับ 9 แทน จะได้โครงสร้างข้อมูลเซตไม่มีส่วนร่วมหลังเรียกใช้ union(6,3) ดังรูปที่ 11.17

i	0	1	2	3	4	5	6	7	8	9
id[i]	0	1	9	9	9	9	9	7	8	9

รูปที่ 11.17 โครงสร้างเซตไม่มีส่วนร่วมหลังเรียกใช้ union(6,3) ด้วยอัลกอริทึมค้นหาแบบเร็ว

(2) อัลกอริทึมยูเนียนแบบเร็ว

อัลกอริทึมยูเนียนแบบเร็ว (Quick-union Algorithm) เป็นวิธีการหนึ่งในการจัดการข้อมูลในโครงสร้างเซตไม่มีส่วนร่วม โดยแต่ละสมาชิกในโครงสร้างเซตจะถูกจัดเก็บด้วยโครงสร้างข้อมูลทรี โดยแต่ละเซตจะเก็บเป็นหนึ่งโครงสร้างทรี และกำหนดให้ตัวแทนของแต่ละเซต คือ ค่าข้อมูลในตำแหน่งรากของทรี โดยจะจัดเก็บข้อมูลด้วยอาร์เรย์ idขนาด N โดยให้ id[i] เก็บ ค่าตำแหน่งของโหนดพ่อของสมาชิกที่ i ซึ่งข้อมูลที่อยู่ในเซตเดียวกันเป็นสมาชิกในโครงสร้างทรีเดียวกัน ดังตัวอย่างโครงสร้างข้อมูลเซตไม่มีส่วนร่วมจำนวน 10 ค่า คือ 0 ถึง 9 ดังรูปที่ 11.18 โดยสมาชิก 5 และ 6 อยู่ในเซตเดียวกันเนื่องจากอยู่ในโครงสร้างทรีเดียวกัน ส่วนสมาชิก 2 3 4 และ 9 อยู่ในเซตเดียวกันเนื่องจากอยู่ในโครงสร้างทรีเดียวกัน ดังตัวอย่างในรูปที่ 11.18 (ก) และมีการจัดเก็บโครงสร้างทรีในอาร์เรย์ในรูปที่ 11.18 (ข)



i	0	1	2	3	4	5	6	7	8	9
id[i]	0	1	9	4	9	6	6	7	8	9

(ก) โครงสร้างทรี (ข) การจัดเก็บโครงสร้างทรีในอาร์เรย์

รูปที่ 11.18 โครงสร้างเซตไม่มีส่วนร่วมที่จัดเก็บด้วยโครงสร้างทรีที่จัดเก็บในอาร์เรย์

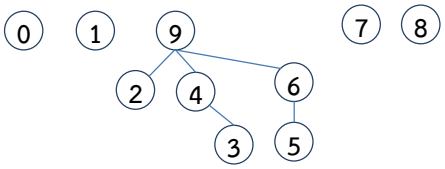
ตัวอย่างคำสั่งภาษาซีในการดำเนินการกับโครงสร้างเซตไม่มีส่วนร่วมที่จัดเก็บด้วยโครงสร้างทรีที่จัดเก็บในอาร์เรย์ มีดังนี้

```
int id[N];
void QuickUnion(){
        for(int i=0; i<id.lenght(); i++)
            id[i] = i;
}
int root(int i){
        while(i !=id[i])
            i=id[i];
            return i;
}
int find(int p, int q){
        return root(p)==root(q);
}
void union(int p, int q){
        int i = root(p);
        int j = root(q);
        id[i]=j;
}
```

- เมื่อเริ่มต้นสร้างโครงสร้างข้อมูลเซตไม่มีส่วนร่วม จะเรียกใช้คำสั่ง QuickUion() สมาชิกแต่ละตัวจะอยู่คนละเซตกัน ดังนั้นจะกำหนดค่า id[i] = i เช่นเดียวกับอัลกอริทึมค้นหาแบบเร็ว

- การดำเนินการค้นหา ก็จะเริ่มที่โหนดของสมาชิกที่ต้องการค้นหาและเปรียบเทียบไปตามเส้นทางของโหนดพ่อจนกระทั่งถึงตำแหน่งรากของทรีด้วยการเรียกใช้คำสั่ง root() หากสมาชิกสองตัวใดมีข้อมูลในตำแหน่งรากเดียวกัน แสดงว่าเป็นข้อมูลในเซตเดียวกัน
- ส่วนการดำเนินการยูเนียนจะค้นหาตำแหน่งรากของสมาชิกแต่ละตัวที่ต้องรวมเซตกัน จากนั้นให้นำรากของสมาชิกไปเป็นโหนดลูกของรากในอีกทรีหนึ่ง

จากรูปที่ 11.18 เมื่อมีการเรียกใช้การดำเนินการ union(6,4) จะต้องทำการหาว่า root[6] มีค่าเท่ากับ 6 และ root[4] มีค่าเท่ากับ 9 ดังนั้นจะทำการเปลี่ยน root[6] ให้มีค่าเท่ากับ 9 เพื่อรวมโครงสร้างทรีของทั้งสองเซตให้เป็นโครงสร้างทรีเดียวกัน (รวมเป็นเซตเดียวกัน) ดังรูปที่ 11.19 (ก) ซึ่งมีผลทำให้มีการจัดเก็บข้อมูลในอาร์เรย์ดังรูปที่ 11.19 (ข)

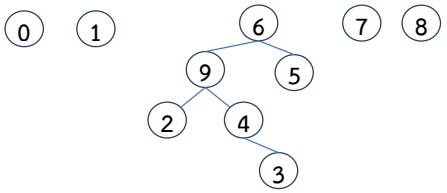


i	0	1	2	3	4	5	6	7	8	9
id[i]	0	1	9	4	9	6	9	7	8	9

(ก) โครงสร้างทรี (ข) การจัดเก็บโครงสร้างทรีในอาร์เรย์

รูปที่ 11.19 โครงสร้างเซตไม่มีส่วนร่วมหลังเรียกใช้ union(4,6) ด้วยอัลกอริทึมยูเนียนแบบเร็ว

อย่างไรก็ตาม จากรูปที่ 11.18 เมื่อมีการเรียกใช้การดำเนินการ union(4,6) แทนจะต้องทำการหาว่า root[4] มีค่าเท่ากับ 9 และ root[6] มีค่าเท่ากับ 6 แล้วจึงทำการเปลี่ยน root[9] ให้มีค่าเท่ากับ 6 เพื่อรวมโครงสร้างทรีของทั้งสองเซตให้เป็นโครงสร้างทรีเดียวกันดังรูปที่ 11.20 (ก) และมีผลให้การจัดเก็บข้อมูลในอาร์เรย์เป็นดังรูปที่ 11.20 (ข) ซึ่งจะพบว่าโครงสร้างทรีที่ได้มีความสูงของทรีมากกว่าการเรียกใช้การดำเนินการ union(6,4)



i	0	1	2	3	4	5	6	7	8	9
id[i]	0	1	9	4	6	6	9	7	8	6

(ก) โครงสร้างทรี (ข) การจัดเก็บโครงสร้างทรีในอาร์เรย์

รูปที่ 11.20 โครงสร้างเซตไม่มีส่วนร่วมหลังเรียกใช้ union(6,4) ด้วยอัลกอริทึมยูเนียนแบบเร็ว

ดังนั้นจะเห็นว่าปัญหาของการดำเนินการยูเนียนกับโครงสร้างข้อมูลเซตไม่มีส่วนร่วมด้วยอัลกอริทึมยูเนียนแบบเร็ว อาจมีผลให้ได้โครงสร้างทรีที่มีความสูงมากขึ้นจนได้ทรีแบบเอียงมากเกินไป ดังตัวอย่างลำดับการทำคำสั่งยูเนียนต่อไปนี้

ตัวอย่างที่ 1

3-4, 4-9, 8-0, 2-3, 5-6, 5-9, 7-3, 4-8, 6-1

ผลของการเรียกใช้การทำยูเนียน 9 ลำดับคำสั่งยูเนียนข้างต้นด้วยอัลกอริทึมยูเนียนแบบเร็ว จะได้โครงสร้างทรีที่สูงเพิ่มขึ้นเรื่อย ๆ จนได้โครงสร้างทรีดังรูปที่ 11.21 (ก)

ตัวอย่างที่ 2

9-4, 4-3, 0-8, 3-2, 6-5, 9-5, 3-7, 8-4, 1-6

ผลของการเรียกใช้การทำยูเนียน 9 ลำดับคำสั่งยูเนียนข้างต้นด้วยอัลกอริทึมยูเนียนแบบเร็ว จะได้โครงสร้างทรีที่สูงเพิ่มขึ้นเรื่อย ๆ จนได้โครงสร้างทรีดังรูปที่ 11.21 (ข) ซึ่งพบว่าจะได้โครงสร้างทรีที่มีความสูงมากเช่นกัน

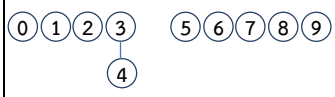
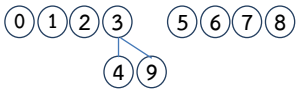
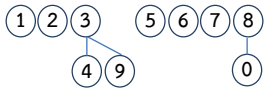
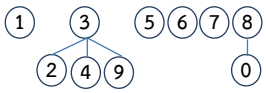
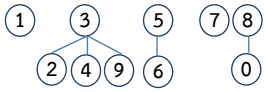
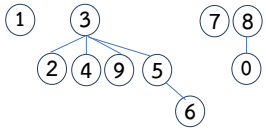
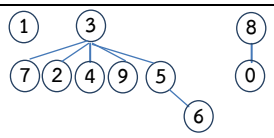
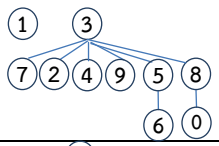
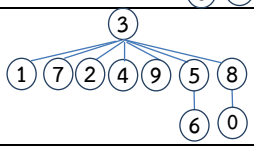
(ก) ตัวอย่างที่ 1 **(ข) ตัวอย่างที่ 2**

รูปที่ 11.21 ตัวอย่างโครงสร้างเซตไม่มีส่วนร่วมด้วยอัลกอริทึมยูเนียนแบบเร็ว

(3) อัลกอริทึมยูเนียนแบบเร็วที่มีการถ่วงน้ำหนัก

จากรูปที่ 11.21 พบว่าการยูเนียนกับโครงสร้างข้อมูลเซตไม่มีส่วนร่วมด้วยอัลกอริทึมยูเนียนแบบเร็ว อาจมีผลให้ ได้โครงสร้างทรีที่มีความสูงเพิ่มขึ้นจน ทำให้เวลาที่ใช้ในการดำเนินการ ค้นหาและยูเนียนมีค่าเทียบเท่า O(n) จึงมีการพัฒนาอัลกอริทึมยูเนียนแบบเร็วที่มีการถ่วงน้ำหนัก (Weighted Quick-union Algorithm) เพื่อลดปัญหาโครงสร้างทรีที่สูงเพิ่มขึ้นเรื่อยๆ โดยปรับวิธีการดำเนินการยูเนียนของอัลกอริทึมยูเนียนแบบเร็ว ให้สามารถเก็บข้อมูลและขนาด ของทรีใหม่ทุกครั้งเมื่อมี การยูเนียน ค่าขนาดของทรีจะถูกใช้ในการ ตรวจสอบ เพื่อกำหนดวิธีการยูเนียน โดยจะ กำหนดให้ทรีที่มีขนาดน้อยกว่าไปเป็นลูกของรากของทรีที่มีขนาดมากกว่า

ตารางที่ 11.7 ตัวอย่างการเรียกใช้คำสั่งยูเนียนของอัลกอริทึมยูเนียนแบบเร็วที่มีการถ่วงน้ำหนัก

ลำดับ	ยูเนียน	โครงสร้างทรี	อาร์เรย์ id	คำอธิบาย
1	-		0 1 2 3 4 5 6 7 8 9	ทุกสมาชิก x อยู่คนละเซตกัน ดังนั้น size(x)=1ทุกค่า x
2	3-4		0 1 2 3 3 5 6 7 8 9	ค่า root(3) คือ 3,root(4) คือ 4 size(3) = 1 และ size(4) = 1 ดังนั้นid[4]=3, size(3)=1+1=2
3	4-9		0 1 2 3 3 5 6 7 8 3	ค่า root(4) คือ 3,root(9) คือ 9 size(3) = 2 และ size(9) = 1 ดังนั้น id[9]=3, size(3)= 2+1 = 3
4	8-0		8 1 2 3 3 5 6 7 8 3	ค่า root(8) คือ 8,root(0) คือ 0size(8)=1และ size(0)=1 ดังนั้น id[0]=8, size(8)=1+1=2
5	2-3		8 1 3 3 3 5 6 7 8 3	ค่า root(2) คือ 2,root(3) คือ 3 size(2)=1 และ size(3)=3 ดังนั้น id[2]=3, size(3)= 3+1 = 4
6	5-6		8 1 3 3 3 5 5 7 8 3	ค่า root(5) คือ 5,root(6) คือ 6 size(5) = 1 และ size(6) = 1 ดังนั้น id[6]=5, size(5)= 1+1 = 2
7	5-9		8 1 3 3 3 3 5 7 8 3	ค่า root(5) คือ 5,root(9) คือ 3 size(5) = 2 และ size(3) = 4 ดังนั้น id[5]=3, size(3)= 4+2 = 6
8	7-3		8 1 3 3 3 3 5 3 8 3	ค่า root(7) คือ 7,root(3) คือ 3 size(7) = 1 และ size(3) = 6 ดังนั้น id[7]=3, size(3)= 6+1 = 7
9	4-8		8 1 3 3 3 3 5 3 3 3	ค่า root(4) คือ 3,root(8) คือ 8 size(3) = 7 และ size(8) = 2 ดังนั้น id[8]=3, size(3)= 7+2 = 9
10	6-1		8 3 3 3 3 3 5 3 3 3	ค่า root(6) คือ 3,root(1) คือ 1size(3)=9และ size(1) = 1 ดังนั้น id[8]=3, size(3)= 9+1 = 10

จากตารางที่ 11.7 พบว่าหลังการดำเนินการยูเนียนครบทั้ง 9 คำสั่ง จะได้โครงสร้างทรีที่ใช้ในการจัดเก็บข้อมูลของเซตไม่มีส่วนร่วมที่มีความสูงของทรีน้อยกว่า โครงสร้างทรีที่ได้หลังการดำเนินการยูเนียนด้วยอัลกอริทึมยูเนียนแบบเร็วในรูปที่ 11.6 ดังนั้นเมื่อเปรียบเทียบเวลาที่ใช้ในการดำเนินการของคำสั่งค้นหาและยูเนียนสำหรับอัลกอริทึมต่างๆ ที่ใช้ในการจัดการข้อมูลโครงสร้างเซต สามารถแสดงได้ดังตารางที่ 11.8 ซึ่งจะพบว่าการดำเนินการต่างๆ กับโครงสร้างข้อมูลที่ใช้อัลกอริทึมค้นหาแบบเร็วจะค้นหาด้วยเวลา O(1) แต่เวลาที่ใช้ในการทำการดำเนินการยูเนียนก็ยังคงเป็น O(n) อยู่ดี ในขณะที่อัลกอริทึมยูเนียนแบบเร็วจะใช้เวลาเท่ากับ O(n) เนื่องจากอาจได้โครงสร้างทรีที่มีความสูงเทียบเท่ากับขนาดของข้อมูลที่จัดเก็บ ส่วนโครงสร้างข้อมูลที่ใช้อัลกอริทึมยูเนียนแบบเร็วที่มีการถ่วงน้ำหนัก จะมีค่าน้อยที่สุดคือ O(log(n)) เนื่องจากเป็นเวลาที่ใช้ในการค้นหารากของทรีในเซตที่จะถูกดำเนินการ

ตัวอย่างคำสั่งในการคำนวณขนาดของทรี

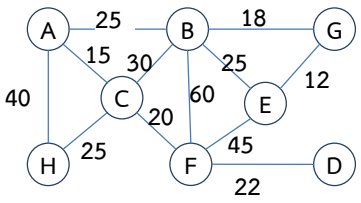
```
if(size[i] < size[j]) {
    id[i] = j;
    size[j] += size[i];
} else {
    id[j] = i;
    size[i] += size[j];
}
```

ตารางที่ 11.8 เปรียบเทียบเวลาในการทำงานแต่ละอัลกอริทึมกับโครงสร้างข้อมูลเซตไม่มีส่วนร่วม

โครงสร้างข้อมูลในแต่ละอัลกอริทึม	**ยูเนียน**	**ค้นหา**
ค้นหาแบบเร็ว	n	1
ยูเนียนแบบเร็ว	n	n
ยูเนียนแบบเร็วที่มีการถ่วงน้ำหนัก	log n	log n

แบบฝึกหัด

1. จากรูปกราฟที่กำหนดให้ให้ตอบคำถามต่อไปนี้



   1.1 จงหาค่าดีกรีของแต่ละจุดยอดในกราฟ และแสดงการจัดเก็บข้อมูลกราฟในลักษณะของแอดจาเซนซีลิสต์
   1.2 จงยกตัวอย่างเส้นทางที่มีวง และมีความยาวของเส้นทางไม่น้อยกว่า 3 มา 2 ตัวอย่าง
   1.3 จงแสดงการจัดเก็บโครงสร้างกราฟด้วยแอดจาเซนซีเมทริกซ์และแอดจาเซนซีลิสต์
   1.4 จงแสดงลำดับการค้นหาจุดยอด E ในกราฟที่กำหนดให้ด้วยวิธีการค้นหาแบบแนวกว้าง และการค้นหาแบบแนวลึกเมื่อกำหนดจุดเริ่มต้นที่จุดยอด A
   1.5 จงแสดงขั้นตอนการหาทรีแบบทอดข้ามที่สั้นที่สุดด้วยอัลกอริทึมของครูสกาล และพริม
   1.6 จงแสดงขั้นตอนการหาเส้นทางที่สั้นที่สุดด้วยวิธีไดก์สตรา เมื่อกำหนดจุดเริ่มต้นที่จุดยอด A
2. จากลำดับการดำเนินการยูเนียนที่กำหนดให้ จงแสดงโครงสร้างทรีที่ใช้ในการจัดเก็บข้อมูลสำหรับโครงสร้างข้อมูล union-and-find ด้วยวิธี Quick Union และ Weighted Quick Union โดยมีลำดับการดำเนินการยูเนียน 5-7, 4-3, 8-0, 2-3, 5-6, 6-1, 7-4, 3-8, 6-9
3. จงเขียนโปรแกรมในการรับข้อมูลกราฟขนาด N แล้วแสดงเส้นทางที่เป็นไปได้ทั้งหมด เมื่อกำหนดจุดยอดต้นทางและจุดยอดปลายทาง
4. จงเขียนโปรแกรมในการรับข้อมูลกราฟขนาด N แล้ว
   4.1 หาเส้นทางที่เป็นไปได้ทั้งหมด เมื่อกำหนดจุดยอดต้นทางและจุดยอดปลายทาง
   4.2 หาเส้นทางที่ยาวที่สุด เมื่อกำหนดจุดยอดต้นทางและจุดยอดปลายทาง
   4.3 หาเส้นทางที่สั้นที่สุด จากจุดยอดต้นทางที่กำหนดไปยังจุดยอดอื่นๆ ในกราฟ
5. จงเขียนโปรแกรมในการรับข้อมูลกราฟขนาด N แล้วทำการตรวจสอบว่ากราฟที่กำหนดให้มี
   5.1 เป็นกราฟแบบไม่ต่อเนื่องหรือไม่
   5.2 เป็นกราฟที่มีอย่างน้อย 1 เส้นทางแบบมีวงหรือไม่